HP 9800 Computer Systems

BASIC Language Reference
for the HP 9826 Computer

HEWLETT PACKARD

BASIC Language Reference for the HP 9826 Computer

Manual Part No. 09826-90055
Microfiche No. 09826-99055

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

November 1981...First Edition

Table of Contents

Keyword Dictionary

Information Provided

This section contains an alphabetical reference to all the keywords currently available with the standard BASIC language system of the 9826. Each entry defines the keyword, shows the proper syntax for its use, gives some example statements, and explains relevant semantic details. A cross reference is provided in the back that groups the keywords into several functional categories.

Above each drawing is a small table indicating the legal uses of the keyword. "Keyboard Executable" means that a properly constructed statement containing that keyword can be typed into the keyboard input line and executed by a press of the EXECUTE key. "Programmable" means that a properly constructed statement containing that keyword can be placed after a line number and stored in a program. Certain non-programmable keywords can be "forced" into a program by sending them to the keyboard buffer with an OUTPUT 2 statement. This is **not** what is meant by "Programmable".

"In an IF...THEN..." means that a properly constructed statement containing that keyword can be placed after "THEN" in a **single-line** IF...THEN statement. Keywords that are prohibited in a single-line IF...THEN are not necessarily prohibited in a multiple-line IF...THEN structure. IF...THEN and FOR...NEXT statements are executed conditionally when they are included in a multiple-line IF...THEN structure. All other prohibited statements (see IF...THEN) are used only during pre-run. Therefore, the action of those statements will not be conditional, even though the IF...THEN wording may make them appear to be conditional.

Syntax Drawings Explained

Statement syntax is represented pictorially. All characters enclosed by a rounded envelope must be entered exactly as shown. Words enclosed by a rectangular box are names of items used in the statement. A description of each item is given either in the table following the drawing, another drawing, or the Glossary. Statement elements are connected by lines. Each line can be followed in only one direction, as indicated by the arrow at the end of the line. Any combination of statement elements that can be generated by following the lines in the proper direction is syntactically correct. An element is optional if there is a valid path around it. Optional items usually have default values. The table or text following the drawing specifies the default value that is used when an optional item is not included in a statement.

Comments may be added to any valid line. A comment is created by placing an exclamation point after a statement or after a line number. The text following the exclamation point may contain any characters in any order.

The drawings do not deal with the proper use of spaces (ASCII blanks). The computer uses spaces, as well as required punctuation, to distinguish the boundaries between various keywords, names, and other items. In general, at least one space is required between a keyword and a name if they are not separated by other punctuation. Spaces cannot be placed in the middle of keywords or other reserved groupings of symbols. Also, keywords are recognized whether they are typed in uppercase or lowercase. Therefore, to use the letters of a keyword as a name, the name entered must contain some mixture of uppercase and lowercase letters. The following are some examples of these guidelines.

Space Between Keywords and Names

The keyword `NEXT` and the variable `Count` are properly entered with a space between them, as in `NEXT Count`. Without the space, the entire group of characters is interpreted as the name `Nextcount`.

No Spaces in Keywords or Reserved Groupings

The keyword `DELSUB` cannot be entered as `DEL SUB`. The array specifier `(*)` cannot be entered as `( * )`. A function call to "A$" must be entered as `FNA$`, not as `FN A $`. The I/O path name "@Meter" must be entered as `@Meter`, not as `@ Meter`. The "exceptions" are keywords that contain spaces, such as `END IF` and `SCRATCH A`.

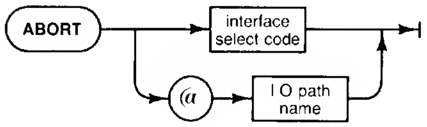
Using Keyword Letters for a Name

Attempting to store the line `IF X=1 THEN END` will generate an error because `END` is a keyword not allowed in an IF...THEN. To create a line label called "End", type `IF X=1 THEN ENd`. This or any other mixture of uppercase and lowercase will prevent the name from being recognized as a keyword.

ABORT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement ceases HP-IB activity. When the 9826 is system controller but not active controller, ABORT causes the 9826 to assume active control.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
I/O path name	name assigned to an HP-IB interface select code	any valid name (see ASSIGN)

Example Statements

```
ABORT 7
IF Stop_code THEN ABORT @Source
```

Summary of Bus Actions

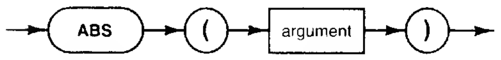
	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	IFC (duration ≥100μsec) REN $\overline{\text{ATN}}$		ATN MTA UNL $\overline{\text{ATN}}$	
Not Active Controller	IFC (duration ≥100 μsec)* REN $\overline{\text{ATN}}$		No Action	

* The IFC message allows a non-active controller (which is the system controller) to become the active controller.

ABS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the absolute value of its argument. The result will be of the same type (REAL or INTEGER) as the argument. (Except for the ABS of the INTEGER −32 768, which causes an error).



Item	Description/Default	Range Restrictions
argument	numeric expression	—

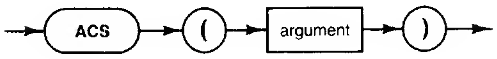
Example Statements

```
Magnitude=ABS(Vector)
PRINT "Value =";ABS(X1)
```

ACS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a cosine equal to the argument. This is the arccosine function.



Item	Description/Default	Range Restrictions
argument	numeric expression	−1 thru +1

Example Statements

```
Angle=ACS(Cosine)
PRINT "Angle =";ACS(X1)
```

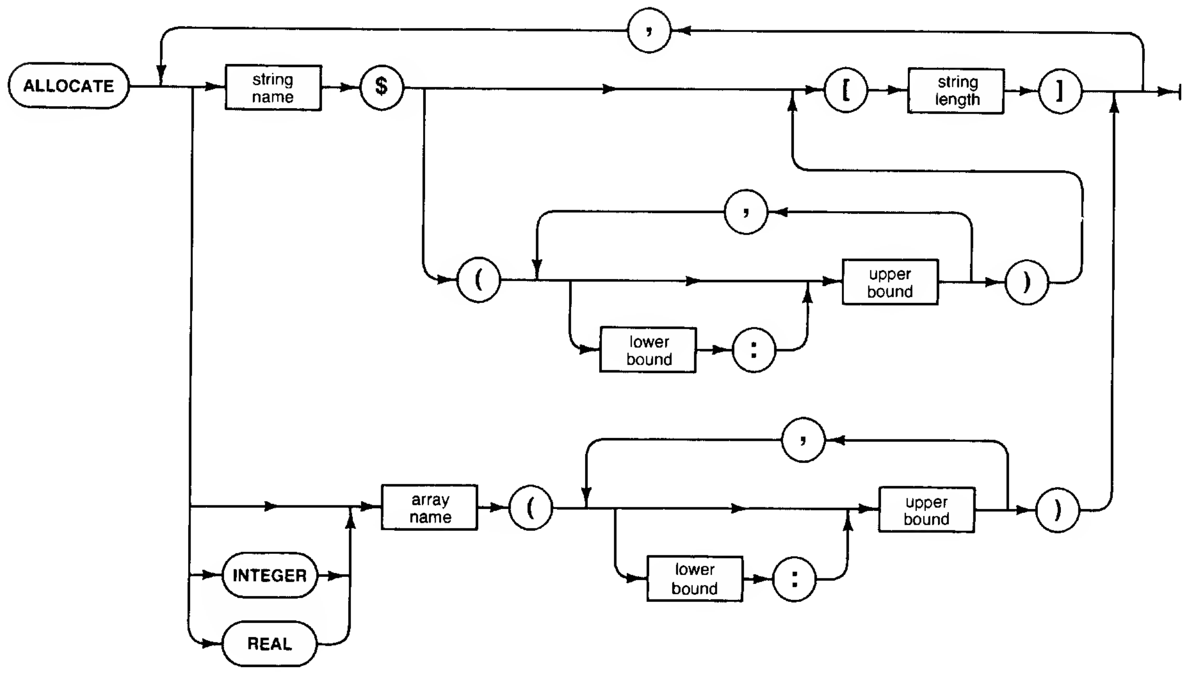
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is 0 thru 180 degrees. If the current angle mode is RAD, the range of the result is 0 thru π radians. The angle mode is radians unless you specify degrees with the DEG statement.

ALLOCATE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement dynamically allocates memory for arrays and string variables during program execution.



Item	Description/Default	Range Restrictions
array name	name of a numeric array	any valid name
lower bound	numeric expression, rounded to an integer; Default = OPTION BASE value (0 or 1)	−32 768 thru +32 767 (see "array" in Glossary)
upper bound	numeric expression, rounded to an integer	−32 768 thru +32 767 (see "array" in Glossary)
string name	name of a string variable	any valid name
string length	numeric expression, rounded to an integer	1 thru 32 767

Example Statements

```
ALLOCATE Temp(Low:High)
ALLOCATE R$[LEN(A$)+1]
```

Semantics

Memory reserved by the ALLOCATE statement can be freed by the DEALLOCATE statement. However, because of the stack discipline used when allocating, the freed memory space does not become available unless all subsequently allocated items are also deallocated. For example, assume that A$ is allocated first, then B$, and finally C$. If a DEALLOCATE A$ statement is executed, the memory space for A$ is not reclaimed until B$ and C$ are deallocated. This same stack is used for setting up ON-event branches, so subsequent ON-event statements can also block the reclamation of deallocated memory.

Variables listed in the ALLOCATE statement can be passed in a parameter list. The variables in an ALLOCATE statement cannot have appeared in COM, DIM, INTEGER or REAL declaration statements or be implicitly declared within the same program context. Numeric variables which are not specified as INTEGER are implicitly declared as REAL. A variable can be re-allocated in its program context only if it has been deallocated and its type and number of dimensions remain the same.

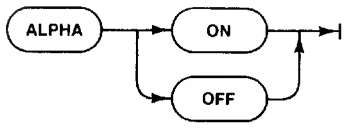
Exiting a subprogram automatically deallocates any memory space allocated within that program context.

ALLOCATE can be executed from the keyboard while a program is running or paused. However, the variable must have been declared in an ALLOCATE statement in the current program context, and the variable must have already been allocated and deallocated.

ALPHA

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement turns the alphanumeric display on or off.



Example Statements

```
ALPHA ON
IF Graph THEN ALPHA OFF
```

Semantics

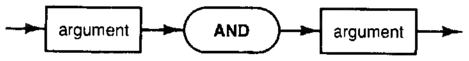
Items sent to the printout area while the alphanumeric display is disabled are placed in the display memory even though they are not visible. Items sent to the keyboard input line, the display line, or the system message line will turn on the alphanumeric display. The alphanumeric and graphic displays can both be on at the same time.

The alphanumeric area is enabled after power-on, RESET and SCRATCH A. Pressing the ALPHA key on the keyboard also enables the alphanumeric display.

AND

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based upon the logical AND of the arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

Example Statements

```
IF Flag AND Test2 THEN Process
Final=Initial AND Valid
```

Semantics

A non-zero value (positive or negative) is treated as a logical 1; only zero is treated as a logical 0.

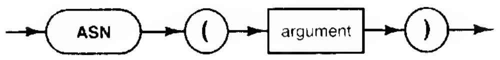
The logical AND is shown in this table:

A B	A AND B
0 0	0
0 1	0
1 0	0
1 1	1

ASN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a sine equal to the argument. This is the arcsine function.



Item	Description/Default	Range Restrictions
argument	numeric expression	−1 thru +1

Example Statements

```
Angle=ASN(Sine)
PRINT "Angle =";ASN(X1)
```

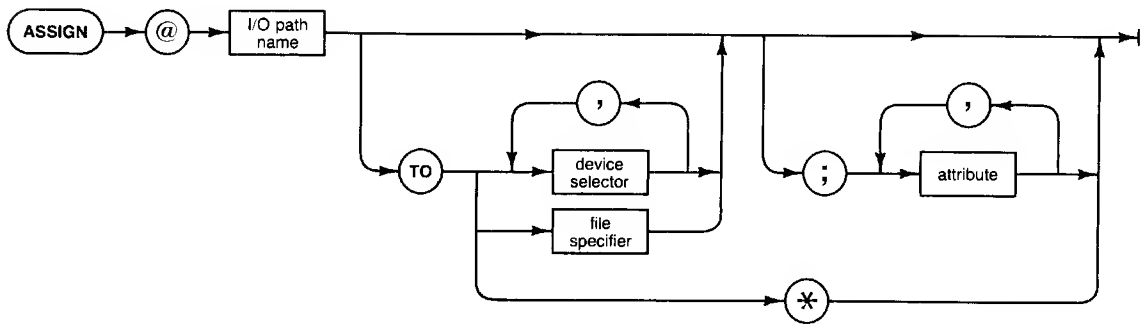
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is −90 thru +90 degrees. If the current angle mode is RAD, the range of the result is $-\pi/2$ thru $+\pi/2$ radians. The angle mode is radians unless you specify degrees with the DEG statement.

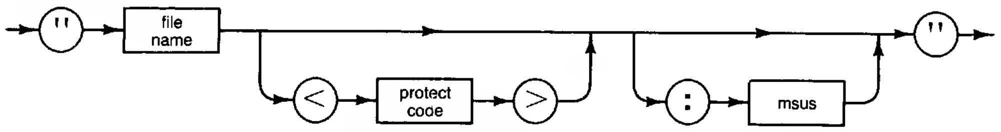
ASSIGN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement assigns an I/O path name and attributes to a device, or group of devices, or a mass storage file.

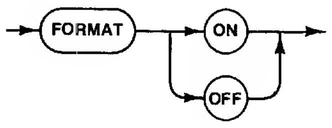


literal form of file specifier:



Protect code not allowed for ASCII files.

attributes:



Item	Description/Default	Range Restrictions
I/O path name	name identifying an I/O path	any valid name
device selector	numeric expression, rounded to an integer	(see Glossary)
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal, first two characters are significant	">" not allowed

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

Example Statements

```
ASSIGN @File TO Name$&Msus$
ASSIGN @Source TO Isc;FORMAT OFF
ASSIGN @Listeners TO 711,712,715
ASSIGN @Dest TO *
```

Semantics

The ASSIGN statement has three primary purposes. Its main purpose is to create an I/O path name and assign that name to an I/O resource and attributes that describe the use of that resource. The statement is also used to change the attributes of an existing I/O path and to close an I/O path.

Associated with an I/O path name is a unique data type that uses about 200 bytes of memory. I/O path names can be placed in COM statements and can be passed by reference as parameters to subprograms. They cannot be evaluated in a numeric or string expression and cannot be passed by value.

Once an I/O path name has been assigned to a resource, OUTPUT, ENTER, STATUS, and CONTROL operations can be directed to that I/O path name. This provides the convenience of re-directing I/O operations in a program by simply changing the appropriate ASSIGN statement. The resource assigned to the I/O path name may be an interface, a device, a group of devices on HP-IB, or a mass storage file.

Specifying FORMAT ON causes items to be output or entered in ASCII format. Specifying FORMAT OFF causes items to be output or entered using internal representation. ASCII files use LIF ASCII format regardless of the FORMAT specified. A FORMAT OFF specification is ignored in an assignment to an ASCII file. If an attribute is not explicitly declared, a default value is assumed. The default attributes are:

Resource	Default Attributes
interface/device	FORMAT ON
ASCII file	(always ASCII format)
BDAT file	FORMAT OFF

Using Devices

I/O path names are assigned to devices by placing the device selector after the keyword TO. For example, `ASSIGN @Display TO 1` creates the I/O path name "@Display" and assigns it to the internal CRT. The statement `ASSIGN @Meters TO 710,711,712` creates the I/O path name "@Meters" and assigns it to a group of three devices on HP-IB. When multiple devices are specified, they must be on the same interface.

When an I/O path name which specifies multiple devices is used in an OUTPUT statement, all devices referred to by the I/O path name receive the data. When an I/O path name which specifies multiple devices is used in an ENTER statement, the first device specified sends the data to the computer and to the rest of the devices. When an I/O path name which specifies multiple HP-IB devices is used in either CLEAR, LOCAL, PPOLL CONFIGURE, PPOLL UNCONFIGURE, REMOTE, or TRIGGER statement, all devices associated with the I/O path name receive the HP-IB message.

A device can have more than one I/O path name associated with it. Each I/O path name can have different attributes, depending upon how the device is used. The specific I/O path name used for an I/O operation determines which set of attributes is used for that operation.

Using Files

Assigning an I/O path name to a file name associates the I/O path with a file on the mass storage media. The mass storage file must be a data file, either ASCII or BDAT. The file must already exist on the media, as ASSIGN does not do an implied CREATE.

ASCII and BDAT files have a position pointer which is associated with each I/O path name. The position pointer identifies the next byte to be written or read, and the value of the position pointer is updated with each ENTER or OUTPUT that uses that I/O path name. The position pointer is reset to the beginning of the file when the file is opened. A file is opened by any ASSIGN statement that includes the file specifier. It is best if a file is open with only one I/O path name at a time.

BDAT files have an additional pointer for end-of-file. The end-of-file value from the media is read when the file is opened. The end-of-file pointer is updated on the media at the following times:

- When the current end-of-file changes.
- When END is specified in an OUTPUT statement directed to the file.
- When a CONTROL statement directed to the I/O path name changes the position of the end-of-file pointer.

Changing Attributes

The attributes of an I/O path may be changed without otherwise disturbing the state of that I/O path or its resource. This is done by deleting the "TO..." clause. For example, `ASSIGN @File;FORMAT OFF` assigns internal format to the I/O path name "@File". If this name were associated with a mass storage file, the pointers would be unaffected. A statement like `ASSIGN @Dvm` restores the default values to all atributes.

Closing I/O Paths

There are a number of ways that I/O paths are closed and the I/O path names are rendered invalid. Closing an I/O path cancels any ON-event actions for that I/O path. I/O path names that are **not** included in a COM statement are closed at the following times:

- When they are explicitly closed; for example, `ASSIGN @File TO *`
- When a currently assigned I/O path name is re-assigned to a resource, the original I/O path is closed before the new one is opened. The re-assignment can be to the same resource or a different resource. No closing occurs when the ASSIGN statement only changes attributes and does not include the "TO..." clause.
- When an I/O path name is a local variable within a subprogram, it is closed when the subprogram is exited by SUBEND, SUBEXIT, RETURN <expression>, or ON <event> RECOVER.
- When any form of SCRATCH statement is executed, any form of STOP occurs, or an END, LOAD, or GET is executed.

I/O path names that are included in a COM statement remain open and valid during a LOAD, GET, STOP, END, or simple SCRATCH. I/O path names in COM are only closed at the following times:

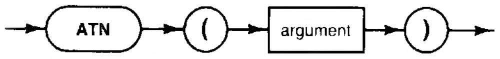
- When they are explicitly closed; for example, `ASSIGN @File TO *`
- When SCRATCH A or SCRATCH C is executed.
- When a LOAD, GET, or EDIT operation brings in a program that has a COM statement that does not exactly match the COM statement containing the open I/O path names.

Additionally, when RESET is pressed, all I/O path names are rendered invalid without going through some of the updating steps that are normally taken to close an I/O path. This is usually not a problem, but there are rare situations which might leave file pointers in the wrong state if their I/O path is closed by a RESET. Explicit closing is preferred and recommended.

ATN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the principal value of the angle which has a tangent equal to the argument. This is the arctangent function.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

Example Statements

```
Angle=ATN(Tangent)
PRINT "Angle =";ATN(X1)
```

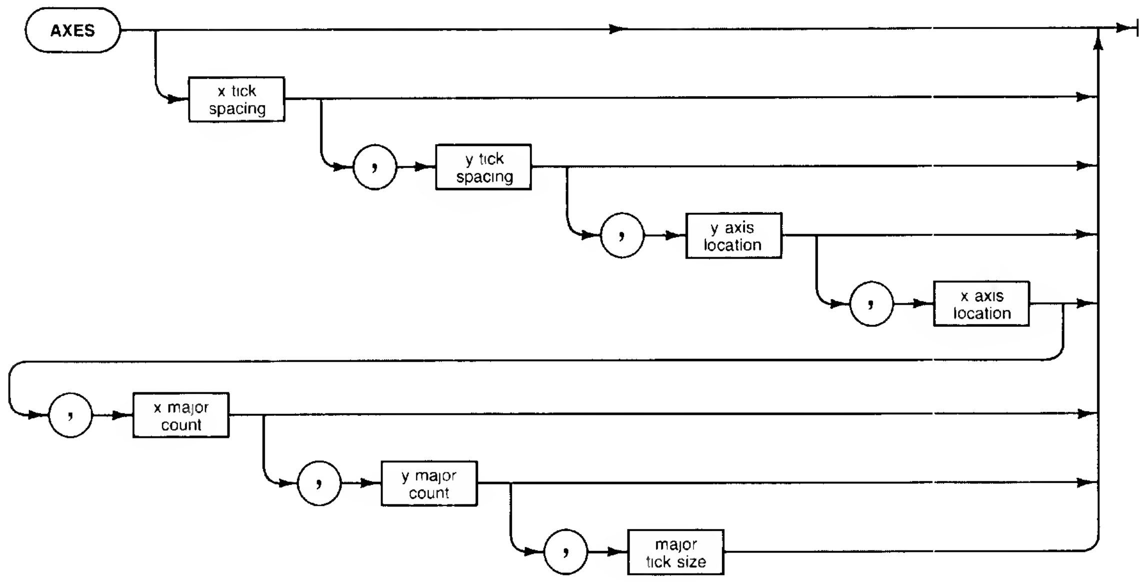
Semantics

The value returned is REAL. If the current angle mode is DEG, the range of the result is -90 thru $+90$ degrees. If the current angle mode is RAD, the range of the result is $-\pi/2$ thru $+\pi/2$ radians. The angle mode is radians unless you specify degrees with the DEG statement.

AXES

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a pair of axes, with optional, equally-spaced tick marks.



Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

Item	Description/Default	Range Restrictions
x tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y axis location	numeric expression specifying the location of the y axis in x-axis units; Default = 0	—
x axis location	numeric expression specifying the location of the x axis in y-axis units; Default = 0	—
x major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
y major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
major tick size	numeric expression in graphic display units; Default = 2	—

Example Statements

```
AXES 10,10
AXES X,Y,Midx,Midy,Maxx/10,Maxy/10
```

Semantics

The axes are drawn so they extend across the soft clip area. The tick marks are symmetric about the axes, but are clipped by the soft clip area. Tick marks are positioned so that a major tick mark coincides with the axis origin, whether or not that intersection is visible. Both axes and tick marks are drawn with the current line type and pen. Minor tick marks are drawn half the size of major tick marks.

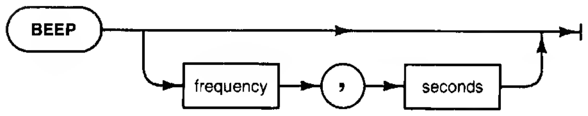
The X and Y tick spacing must not generate more than 32 768 tick marks in the clip area (including the axis), or error 20 will be generated.

If either axis lies outside the current clip area, that axis and its associated tick marks are not drawn.

BEEP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement produces one of 63 audible tones.



Item	Description/Default	Range Restrictions	Recommended Range
frequency	numeric expression, rounded to the nearest tone; Default = 1220.7 Hz	—	81 thru 5127
seconds	numeric expression, rounded to the nearest hundredth; Default = 0.2	—	0.01 thru 2.55

Example Statements

```
BEEP 81.38*Tone,.5
BEEP
```

Semantics

The frequency and duration of the tone are subject to the resolution of the built in tone generator. The frequency specified is rounded to the nearest frequency shown below. For example, any specified frequency from 40.7 to 122.08 produces a beep of 81.38 Hz. If the frequency specified is larger than 5086.25, a tone of 5126.94 is produced. If it is less than 40.69, it is considered to be a 0 and no tone is produced. The following list shows the frequencies available:

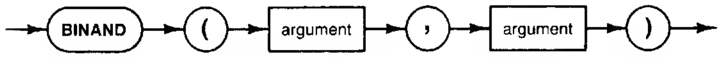
0.0	651.04	1302.08	1953.12	2604.16	3255.20	3906.24	4557.28
81.38	732.42	1383.46	2034.50	2685.54	3336.58	3987.62	4638.66
162.76	813.80	1464.84	2115.88	2766.92	3417.96	4069.00	4720.04
244.14	895.18	1546.22	2197.26	2848.30	3499.34	4150.38	4801.42
325.52	976.56	1627.60	2278.64	2929.68	3580.72	4231.76	4882.80
406.90	1057.94	1708.98	2360.02	3011.06	3662.10	4313.14	4964.18
488.28	1139.32	1790.36	2441.40	3092.44	3743.48	4394.52	5045.56
569.66	1220.70	1871.74	2522.78	3173.82	3824.86	4475.90	5126.94

The resolution of the seconds parameter is .01 seconds. Any duration shorter than .005 seconds is treated as near zero. Any duration longer than 2.55 seconds is treated as 2.55 seconds.

BINAND

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of the bit-by-bit complement of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Low_bits=BINAND(Byte,15)
IF BINAND(Stat,3) THEN Bit_set
```

Semantics

The argument for this function is represented as a 16-bit two's-complement integer. Each bit in the representation of the argument is complemented, and the resulting integer is returned.

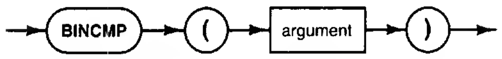
For example, the complement of −9:

$$\begin{array}{rl} & \text{bit 15} \qquad\qquad\qquad \text{bit 0} \\ -9 = & \underline{11111111\ 11110111} \\ & 00000000\ 00001000\ = 8 \end{array}$$

BINCMP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit logical-and of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
True=BINCMP(Inverse)
PRINT X,BINCMP(X)
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is anded with the corresponding bit in the other argument. The results of all the ands are used to construct the integer which is returned.

For example, the statement `Ctrl_word=BINAND(Ctrl_word,-9)` clears bit 3 of Ctrl_word without changing any other bits.

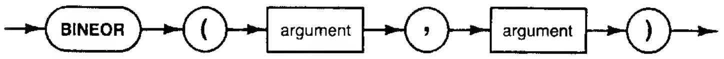
```
            bit 15              bit 0

    12 = 00000000 00001100   old Ctrl_word
    -9 = 11111111 11110111   mask to clear bit 3
         -----------------
     4 = 00000000 00000100   new Ctrl_word
```

BINEOR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit exclusive-or of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Toggle=BINEOR(Toggle,1)
True_byte=BINEOR(Inverse_byte,255)
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is exclusively ored with the corresponding bit in the other argument. The results of all the exclusive ors are used to construct the integer which is returned.

For example, the statement `Ctrl_word=BINEOR(Ctrl_word,4)` inverts bit 2 of Ctrl_word without changing any other bits.

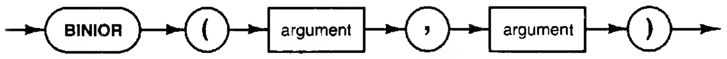
```
         bit 15               bit 0

    12 = 00000000 00001100   old Ctrl_word
     4 = 00000000 00000100   mask to invert bit 2
         -----------------
     8 = 00000000 00001000   new Ctrl_word
```

BINIOR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the value of a bit-by-bit inclusive-or of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
Bits_set=BINIOR(Value1,Value2)
Top_on=BINIOR(All_bits,2^15)
```

Semantics

The arguments for this function are represented as 16-bit two's-complement integers. Each bit in an argument is inclusively ored with the corresponding bit in the other argument. The results of all the inclusive ors are used to construct the integer which is returned.

For example, the statement `Ctrl_word=BINIOR(Ctrl_word,6)` sets bits 1 & 2 of Ctrl_word without changing any other bits.

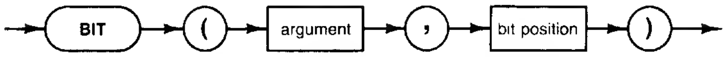
```
        bit 15                bit 0

   19 = 00000000 00010011   old Ctrl_word
    6 = 00000000 00000110   mask to set bits 1 & 2
        -----------------
   23 = 00000000 00010111   new Ctrl_word
```

BIT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a 1 or 0 representing the value of the specified bit of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression, rounded to an integer	−32 768 thru +32 767
bit position	numeric expression, rounded to an integer	0 thru 15

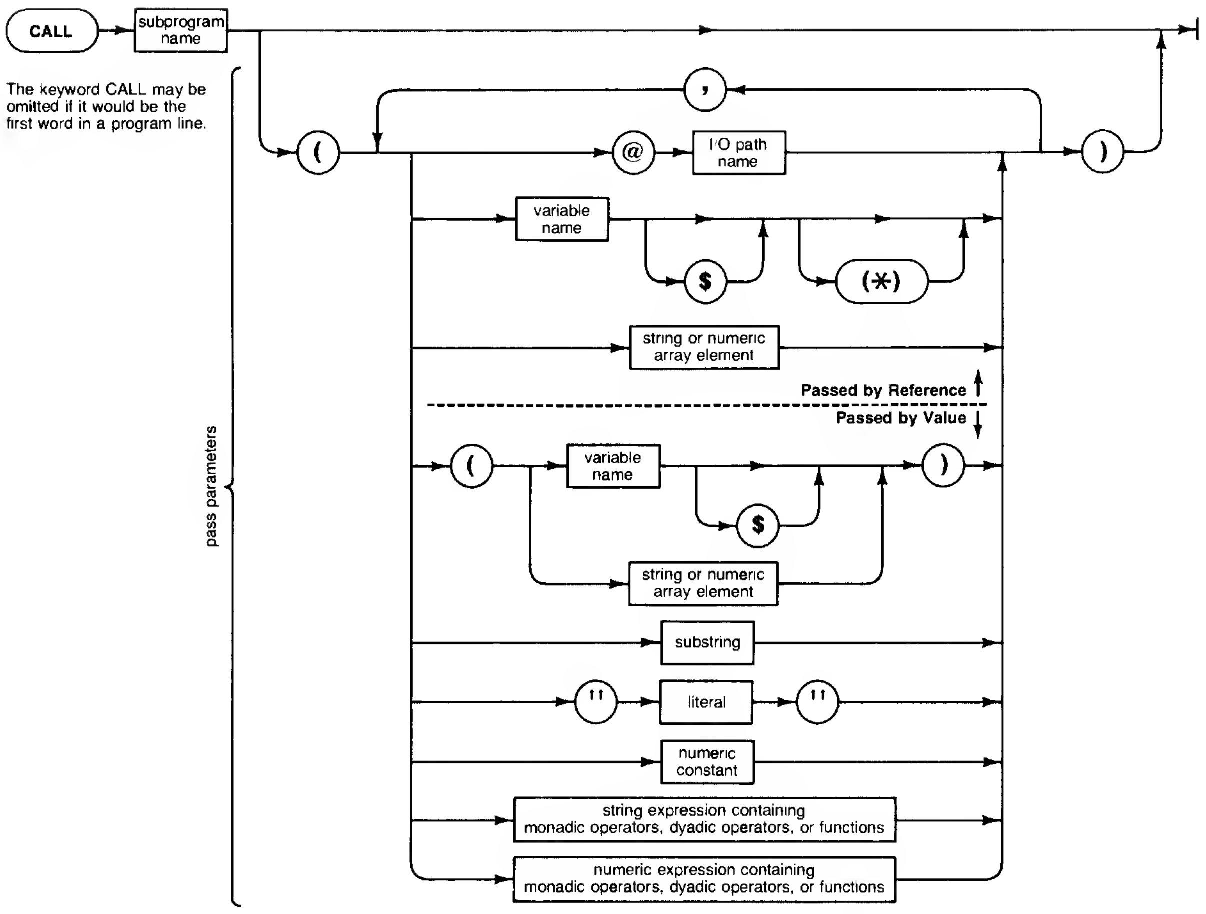
Example Statements

```
Flag=BIT(Info,0)
IF BIT(Word,Test) THEN PRINT "Bit #";Test;"is set"
```

Semantics

The argument for this function is represented as a 16-bit two's-complement integer. Bit 0 is the least-significant bit and bit 15 is the most-significant bit. The following example reads the controller status register of the internal HP-IB and takes a branch to "Active" if the interface is currently the active controller.

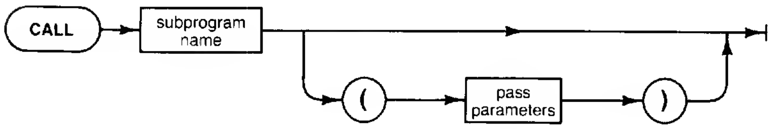
```
100  STATUS 7,3;S            ! Reg 3 = control status
110  IF BIT(S,6) THEN Active  ! Bit 6 = active control
```


CALL
subprogram name
The keyword CALL may be omitted if it would be the first word in a program line.
pass parameters
,
(
@
I/O path name
)
variable name
$
(*)
string or numeric array element
Passed by Reference
Passed by Value
(
variable name
$
)
string or numeric array element
substring
"
literal
"
numeric constant
string expression containing monadic operators, dyadic operators, or functions
numeric expression containing monadic operators, dyadic operators, or functions

CALL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to the specified SUB subprogram and may pass items to the subprogram. SUB subprograms are created with the SUB statement.



Item	Description/Default	Range Restrictions
subprogram name	name of the SUB subprogram to be called	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
variable name	name of a string or numeric variable	any valid name
substring	string expression containing substring notation	(see Glossary)
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—

Example Statements

```
CALL Process(Ref,(Value),@Path)
CALL Transform(Array(*))
IF Flag THEN CALL Special
```

Semantics

A subprogram may be invoked by a stored program line, or by a statement executed from the keyboard. Invoking a subprogram changes the program context. Subprograms may be invoked recursively. The keyword CALL may be omitted if it would be the first word in a program line. However, the keyword CALL is required in all other instances (such as a CALL from the keyboard and a CALL in an IF...THEN... statement).

The pass parameters must be of the same type (numeric, string, or I/O path name) as the corresponding parameters in the SUB statement. Numeric values passed by value are converted to the numeric type (REAL or INTEGER) of the corresponding formal parameter. Variables passed by reference must match the corresponding parameter in the SUB statement exactly. An entire array may be passed by reference by using the asterisk specifier.

If there is more than one subprogram with the same name, the lowest-numbered subprogram is invoked by a CALL.

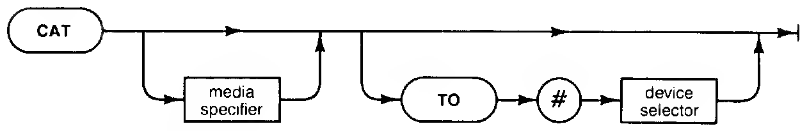
Program execution generally resumes at the line following the subprogram CALL. However, if the subprogram is invoked by an event-initiated branch (ON END, ON ERROR, ON INTR, ON KEY, ON KNOB, or ON TIMEOUT), program execution resumes at the point at which the event-initiated branch was permitted.

When CALL is executed from the keyboard, the current state of the computer determines the computer's state when the subprogram executes a STOP. If the computer was paused or stopped when CALL was executed, its state does not change. If the computer was running when the CALL was executed, the program pauses at the program line which was interrupted by the CALL for the subprogram, and resumes execution at that point after the subprogram is exited.

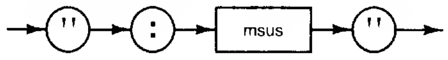
CAT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lists the contents of the mass storage media's directory.



literal form of media specifier:



Item	Description/Default	Range Restrictions
media specifier	string expression	(see drawing)

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

device selector	numeric expression, rounded to an integer; Default = PRINTER IS device	(see Glossary)

Example Statements

```
CAT
CAT TO #701
CAT ":INTERNAL"
```

Semantics

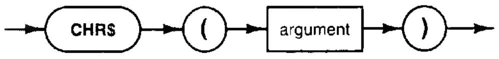
A directory entry is listed for each file on the media. The catalog shows the name of each file, whether or not it is protected, the file's type and length, the number of bytes per logical record, and the starting location (address) of the file on the media.

A protected file has an asterisk in the PRO column entry. The types recognized in BASIC are ASCII, BDAT (BASIC data), BIN (binary program), PROG (BASIC program), or SYSTM (language system). An ID number is listed for any unrecognized file types.

CHR$

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a numeric value into an ASCII character. The low order byte of the 16-bit integer representation of the argument is used; the high order byte is ignored. A table of ASCII characters and their decimal equivalent values may be found in the back of this book.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	0 thru 255

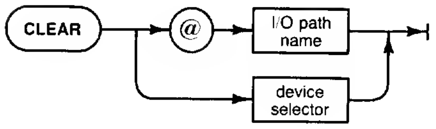
Example Statements

```
A$[Marker;1]=CHR$(Digit+128)
Esc$=CHR$(27)
```

CLEAR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows the active controller to put HP-IB devices into a defined device-dependent state.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
CLEAR 7
CLEAR Isc+Address
CLEAR @Source
```

Semantics

The computer must be the active controller to execute this statement. When primary addresses are specified, the bus is reconfigured and the SDC (Selected Device Clear) message is sent to all devices which are addressed by the LAG message.

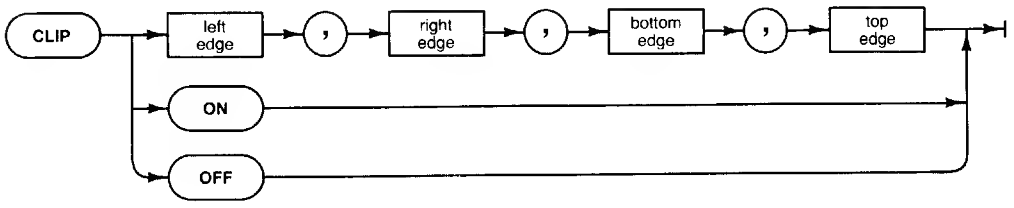
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN DCL	ATN MTA UNL LAG SDC	ATN DCL	ATN MTA UNL LAG SDC
Not Active Controller	Error			

CLIP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement redefines the soft clip area and enables or disables the soft clip limits.



Item	Description/Default	Range Restrictions
left edge	numeric expression	—
right edge	numeric expression	—
bottom edge	numeric expression	—
top edge	numeric expression	—

Example Statements

```
CLIP Left,Right,0,100
CLIP OFF
```

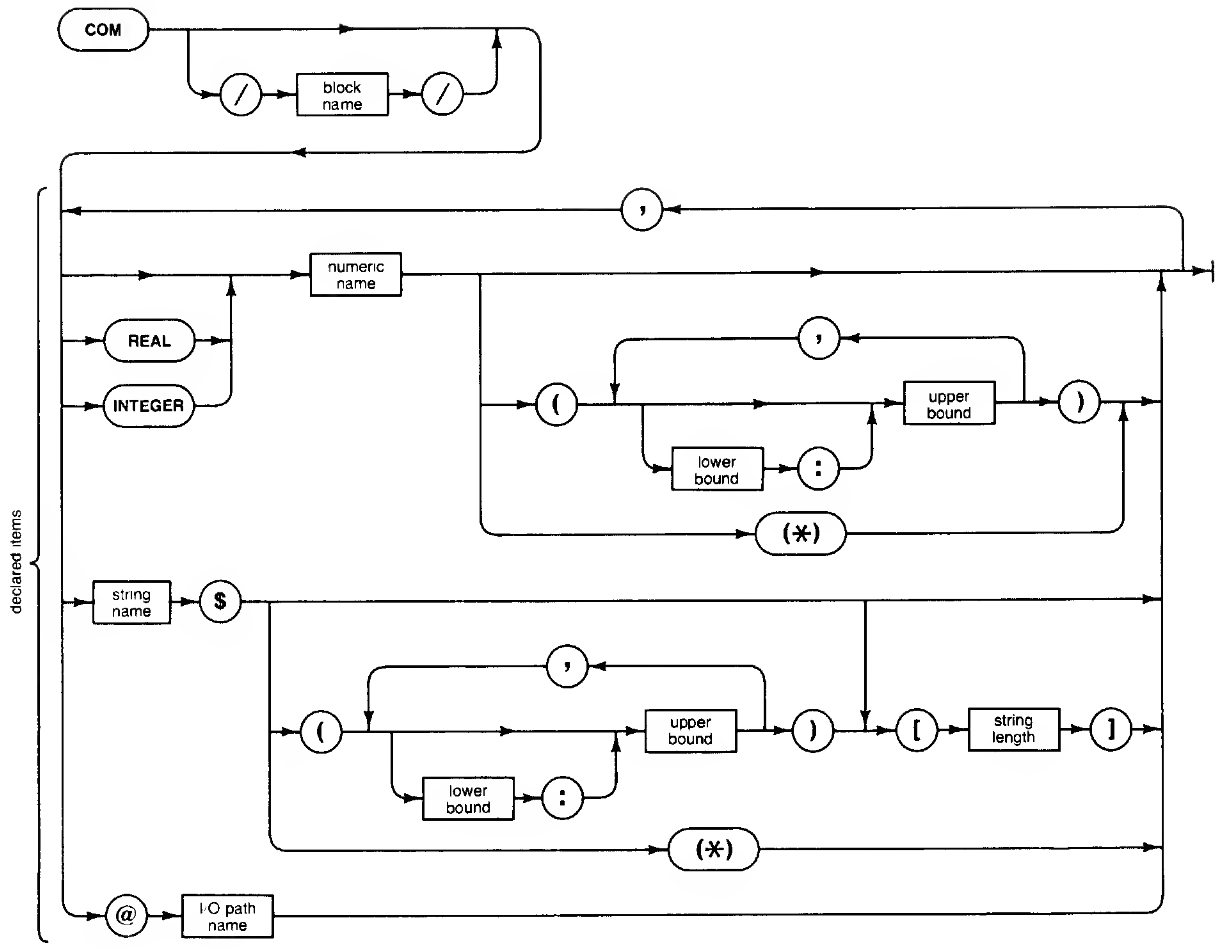
Semantics

Executing CLIP with parameters allows the soft clip area to be changed from the boundary set by PLOTTER IS and VIEWPORT to the soft clip limits. If CLIP is not executed, the area most recently defined by either VIEWPORT or the PLOTTER IS statement is the clipping area. All plotted points, lines or labels are clipped at this boundary.

The hard clip area is specified by the PLOTTER IS statement. The soft clip area is specified by the VIEWPORT and CLIP statements. CLIP ON sets the soft clip boundaries to the last specified CLIP or VIEWPORT boundaries, or to the hard clip boundaries if no CLIP or VIEWPORT has been executed. CLIP OFF sets the soft clip boundaries to the hard clip limits.

CMD

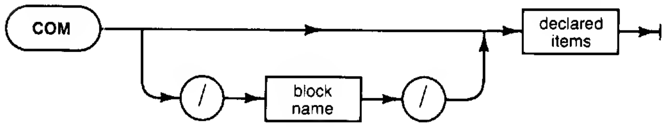
See the SEND statement.


COM
/
block name
/
,
numeric name
REAL
INTEGER
,
(
upper bound
)
lower bound
:
(*)
declared items
string name
$
,
(
upper bound
)
[
string length
]
lower bound
:
(*)
@
I/O path name

COM

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement dimensions and reserves memory for variables in a special "common" memory area so more than one program context can access the variables.



Item	Description/Default	Range Restrictions
block name	name identifying a labeled COM area	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or 1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)
string length	integer constant	1 thru 32 767
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)

Example Statements

```
COM X,Y,Z
COM /Graph/ Title$,@Device,INTEGER Points(*)
COM INTEGER I,J,REAL Array(-128:127)
```

Semantics

Storage for COM is allocated at prerun time in an area of memory which is separate from the data storage used for program contexts. This reserved portion of memory remains allocated until SCRATCH A or SCRATCH C is executed. Changing the definition of the COM space is accomplished by a full program prerun. This can be done by:

- Pressing the RUN or STEP key when no program is running.
- Executing a RUN command when no program is running.
- Executing any GET or LOAD from a program.
- Executing a GET or LOAD command that tells program execution to to begin.

When COM allocation is performed at prerun, the new program's COM area is compared against the COM area currently in memory. Where the two areas agree exactly in type, size, and shape, the COM area is preserved. Any variable values are left intact. All other COM areas are rendered undefined, and their storage area is not recovered by the computer. New COM variables are initialized at prerun: numeric variables to 0, string variables to the null string.

Each context may have as many COM statements as needed (within the limits of computer memory), and COM statements may be interspersed between other statements. If there is an OPTION BASE statement in the context, it must appear before the COM statement. COM variables do not have to have the same names in different contexts. Formal parameters of subprograms are not allowed in COM statements. A COM mismatch between contexts causes an error.

If a COM area requires more than one statement to describe its contents, COM statements defining that block may not be intermixed with COM statements defining other COM areas.

Numeric variables in a COM list can have their type specified as either REAL or INTEGER. Specifying a variable type implies that all variables which follow in the list are of the same type. The type remains in effect until another type is specified. String variables and I/O path names are considered a type of variable and change the specified type. Numeric variables are assumed to be REAL unless their type has been changed to INTEGER.

COM statements (blank or labeled) in different contexts which refer to an array or string must specify it to be of the same size and shape. The lowest-numbered COM statement containing an array or string name must explicitly specify the subscript bounds and/or string length. Subsequent COM statements can reference a string by name or an array by using an asterisk specifier.

No array can have more than six dimensions. The total number of elements is limited by the computer's memory size. The lower bound value must be less than or equal to the upper bound value. The default lower bound is specified by the OPTION BASE statement.

Any LOADSUB which attempts to define or change COM areas while a program is running generates error 145.

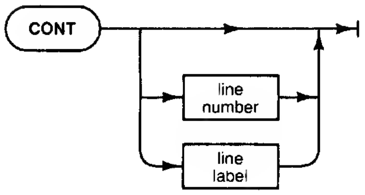
Unlabeled or Blank COM
Blank COM does not contain a block name in its declaration. Blank COM (if it is used) must be created in a main context. The main program can contain any number of blank COM statements. Blank COM areas can be accessed by subprograms, if the COM statements in the subprograms agree in type and shape with the main program COM statements.

Labeled COM
Labeled COM contains a name for the COM area in its declaration. Memory is allocated for labeled COM at prerun time according to the lowest-numbered occurrence of the labeled COM statement. Each context which contains a labeled COM statement with the same label refers to the same labeled COM block.

CONT

Keyboard Executable	Yes
Programmable	No

This command resumes execution of a paused program at the specified line.



Item	Description/Default	Range Restrictions
line number	integer constant identifying a program line; Default = next program line	1 thru 32 766
line label	name identifying a program line	any valid name

Example Statements

```
CONT 550
CONT Sort
```

Semantics

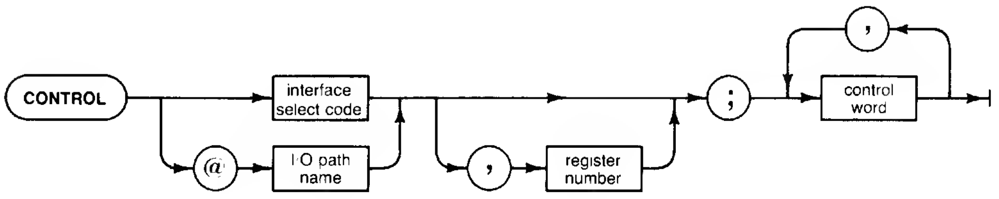
Continue can be executed by pressing the CONTINUE key or by typing a CONT command and pressing EXECUTE. Variables retain their current values whenever CONT is executed. CONT causes the program to resume execution at the next statement which would have occurred, unless a line is specified.

When a line label is specified, program execution resumes at the specified line, provided that the line is in either the main program or the current subprogram. If a line number is specified, program execution resumes at the specified line, provided that the line is in the current program context. If there is no line in the current context with the specified line number, program execution resumes at the next higher-numbered line. If the specified line label does not exist in the proper context, an error results.

CONTROL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends control information to an interface or to the internal table associated with an I/O path name.



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	numeric expression, rounded to an integer	1 thru 31	—
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)	—
register number	numeric expression, rounded to an integer; Default = 0	interface dependent	—
control word	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	0 thru 65 535 (interface dependent)

Example Statements

```
CONTROL @Rand_file,7;File_length
CONTROL 1;Row,Column
CONTROL 7,3;29
```

When the Destination is an I/O Path Name

The only time CONTROL is allowed to an I/O path name is when the I/O path name is assigned to a BDAT file. I/O path names have an association table that can be thought of as a set of registers. Control words are written to the association table, starting with the specified "register" and continuing in turn through the remaining "registers" until all control words are used. The number of control words must not exceed the number of "registers" available. The accessible "registers" for a BDAT file are:

"Register" Number	Contents
5	current record
6	byte within current record
7	EOF record
8	byte within EOF record

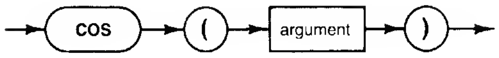
When the Destination is an Interface

Control words are written to the interface registers, starting with the specified register number, and continuing in turn through the remaining registers until all the control words are used. The number of control words must not exceed the number of registers available. Register assignments can be found in the Interface Registers section at the back of the book.

COS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the cosine of the argument. The range of the returned real value is −1 thru +1.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 1.708 312 772 2 E+10 deg. or in radians: 2.981 568 244 292 04 E+8

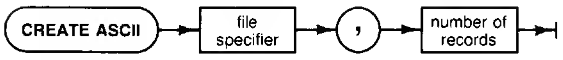
Example Statements

```
Cosine=COS(Angle)
PRINT COS(X+45)
```

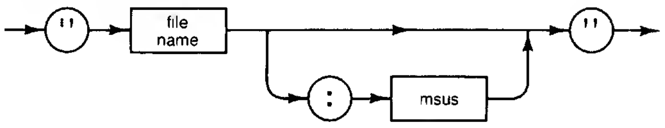
CREATE ASCII

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file on the mass storage media.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
msus	literal; Default = MASS STORAGE IS device	INTERNAL
number of records	numeric expression, rounded to an integer	1 thru $2^{31} - 1$

Example Statements

```
CREATE ASCII "TEXT",100
CREATE ASCII Name$&":INTERNAL",Length
```

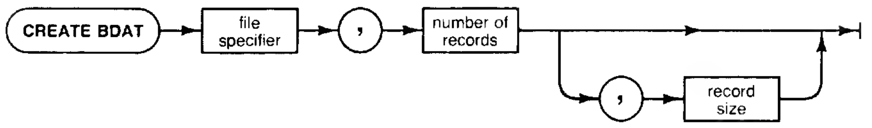
Semantics

CREATE ASCII creates a new ASCII file and directory entry on the mass storage media. CREATE ASCII does not open the file. Opening of files is done by the ASSIGN statement. The records of an ASCII file have a fixed length of 256 bytes. In the event of an error, no directory entry is made and the file is not created.

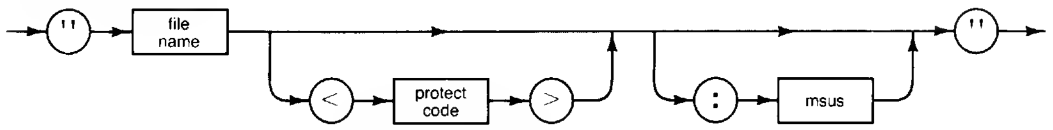
CREATE BDAT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a BDAT file on the mass storage media.



literal form of file specifier



protect code is ignored for ASCII files

Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal, first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL
number of records	numeric expression, rounded to an integer	1 thru $2^{31}-256$
record size	numeric expression, rounded to next even integer (except 1). Specifies bytes/record. Default = 256	1 thru 65 534

Example Statements

```
CREATE BDAT "George",48
CREATE BDAT "Special<PC>",Length,128
CREATE BDAT Name$&Msus$,Bytes,1
```

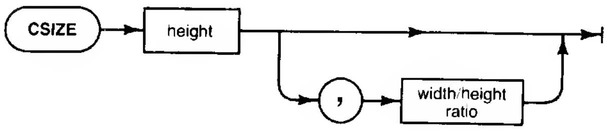
Semantics

CREATE BDAT creates a new BDAT file and directory entry on the mass storage media. CREATE BDAT does not open the file. Opening of files is done by the ASSIGN statement. If a protect code is included after the file name, the first two characters become the protect code of the file. In the event of an error, no directory entry is made and the file is not created. A sector is created at the beginning of the file for system use. This sector cannot be accessed by BASIC programs.

CSIZE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sets the size and aspect (width/height) ratio of the character cell used by the LABEL statement.



Item	Description/Default	Range Restrictions
height	numeric expression; Default = 5	—
width/height ratio	numeric expression; Default = 0.6	—

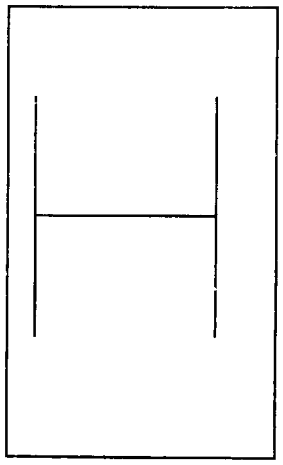
Example Statements

```
CSIZE 10
CSIZE Size,Width
```

Semantics

At power-on, RESET, and GINIT, the height is 5 graphic-display-units (GDUs), and the aspect ratio is 0.6 (width = 3 GDUs, or 0.6 × 5 GDUs). A negative number for either parameter inverts the character along the associated dimension. The drawing below shows the relation between the character cell and a character.

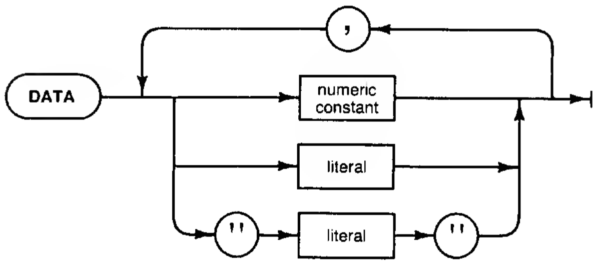
Character in a Character Cell



DATA

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement contains data which can be read by READ statements. (For information about DATA as a secondary keyword, see the SEND statement.)



Item	Description/Default	Range Restrictions
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—

Example Statements

```
DATA 1,1.414,1.732,2
DATA word1,word2,word3
DATA "ex-point(!)","quote("")","comma(,)"
```

Semantics

A program or subprogram may contain any number of DATA statements at any locations. When a program is run, the first item in the lowest numbered DATA statement is read by the first READ statement encountered. When a subprogram is called, the location of the next item to be read in the calling context is remembered in anticipation of returning from the subprogram. Within the subprogram, the first item read is the first item in the lowest numbered DATA statement within the subprogram. When program execution returns to the calling context, the READ operations pick up where they left off in the DATA items.

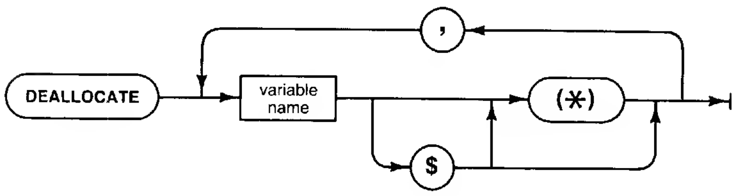
A numeric constant must be read into a variable which can store the value it represents. The computer cannot determine the intent of the programmer; although attempting to read a string value into a numeric variable will generate an error, numeric constants will be read into string variables with no complaint. In fact, the computer considers the contents of all DATA statements to be literals, and processes items to be read into numeric variables with a VAL function, which can result in error 32 if the numeric data is not of the proper form (see VAL).

Unquoted literals may not contain quote marks (which delimit strings), commas (which delimit data items), or exclamation marks (which indicate the start of a comment). Leading and trailing blanks are deleted from unquoted literals. Enclosing a literal in quote marks enables you to include any punctuation you wish, including quote marks, which are represented by a set of two quote marks.

DEALLOCATE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deallocates memory space reserved by the ALLOCATE statement.



Item	Description/Default	Range Restrictions
variable name	name of an array or string variable	any valid name

Example Statements

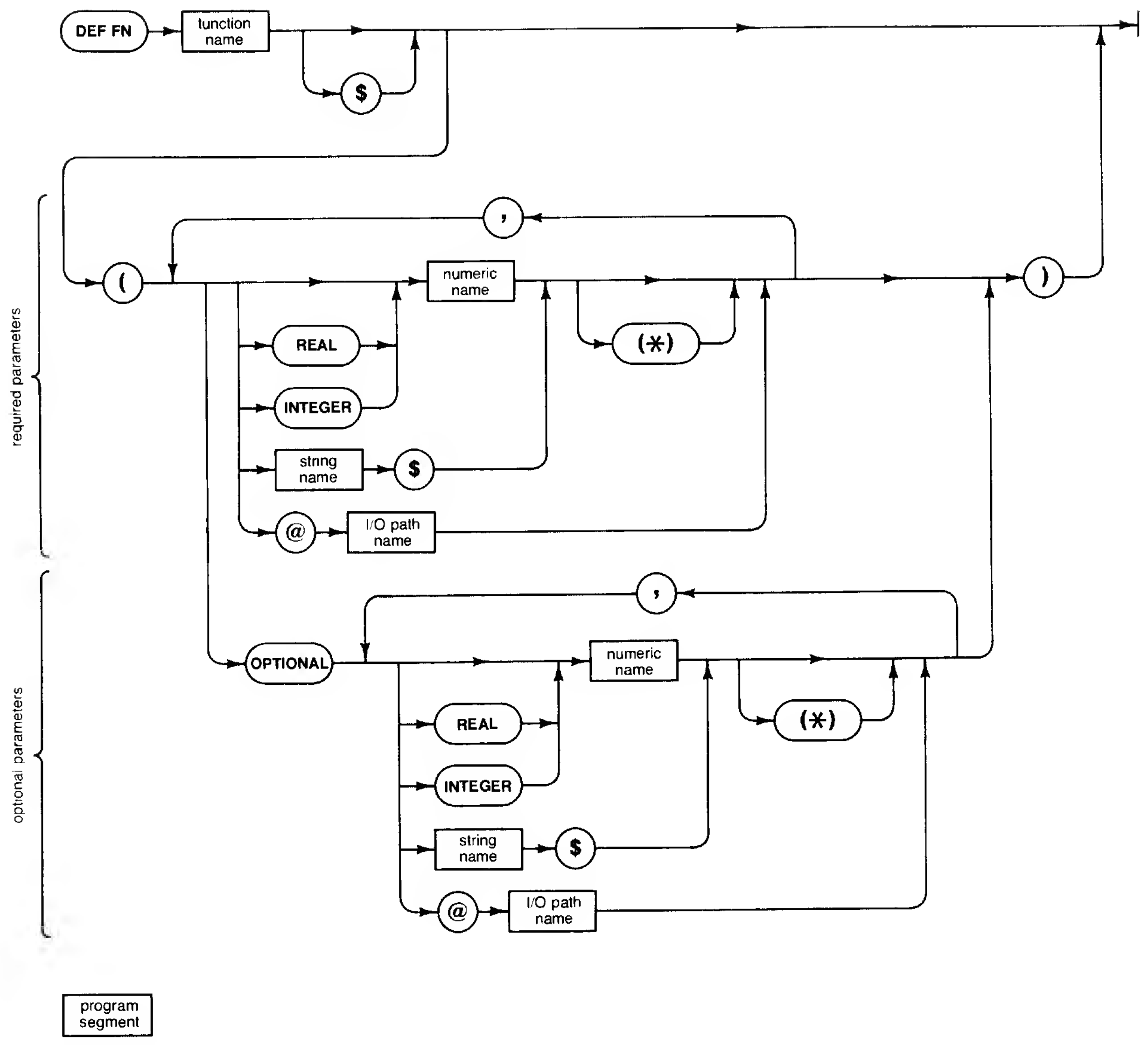
```
DEALLOCATE A$,B$,C$
DEALLOCATE Array(*)
```

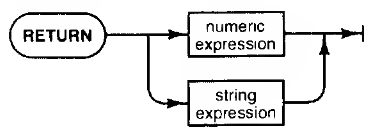
Semantics

Memory space reserved by ALLOCATE exists in the same section of memory as that used by ON-event statements. Since entries in this area are "stacked" as they come in, space for variables which have been DEALLOCATED may not be available immediately. It will not be available until all the space "above it" is freed. This includes variables allocated after it, as well as ON-event entries. Exiting a subprogram automatically deallocates space for variables which were allocated in that subprogram.

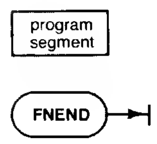
Strings and arrays must be deallocated completely. Deallocation of an array is requested by the (*) specifier.

Attempting to DEALLOCATE a variable which is not currently allocated in the current context results in an error. When DEALLOCATE is executed from the keyboard, deallocation occurs within the current context.





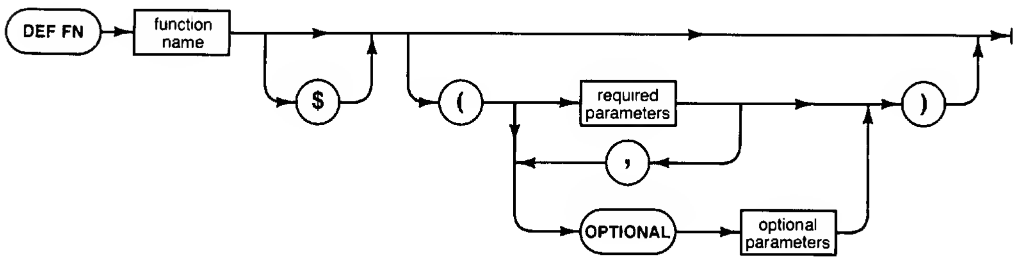
Note: A user-defined function may contain any number of RETURN statements.



DEF FN

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement indicates the beginning of a function subprogram. It also indicates whether the function is string or numeric and defines the formal parameter list.



Item	Description/Default	Range Restrictions
function name	name of the user-defined function	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Statements

```
DEF FNTrim$(String$)
DEF FNTransform(@Printer,INTEGER Array(*),OPTIONAL Text$)
```

Semantics

User-defined functions must appear after the main program. The first line of the function must be a DEF FN statement. The last line must be an FNEND statement. Comments after the FNEND are considered to be part of the function.

Parameters to the left of the keyword OPTIONAL are required and must be supplied whenever the user-defined function is invoked (see FN). Parameters to the right of OPTIONAL are optional, and only need to be supplied if they are needed for a specific operation. Optional parameters are associated from left to right with any remaining pass parameters until the pass parameter list is exhausted. An error is generated if the function tries to use an optional parameter which did not have a value passed to it. The function NPAR can be used to determine the number of parameters supplied by the function call.

Parameters in the formal parameter list may not be duplicated in COM statements. A user-defined function may not contain any SUB statements or DEF FN statements. User-defined functions can be called recursively and may contain local variables. A unique labeled COM must be used if the local variables are to preserve their values between invocations of the user-defined function.

The RETURN <expression> statement is important in a user-defined function. If the program actually encounters an FNEND during execution (which can only happen if the RETURN is missing or misplaced), error 5 is generated. The <expression> in the RETURN statement must be numeric for numeric functions, and string for string functions. A string function is indicated by the dollar sign suffix on the function name.

The purpose of a user-defined function is to compute a single value. While it is possible to alter variables passed by reference and variables in COM, this can produce undesirable side effects, and should be avoided. If more than one value needs to be passed back to the program, SUB subprograms should be used.

DEG

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects degrees as the unit of measure for expressing angles.

Semantics

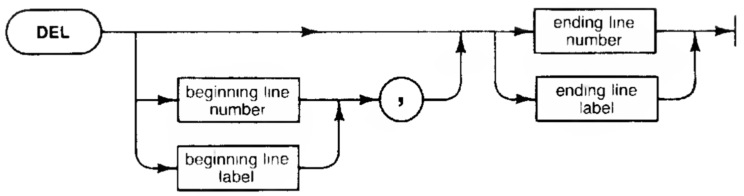
All functions which return an angle will return an angle in degrees. All operations with parameters representing angles will interpret the angle in degrees.

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context. If no angle mode is specified in a program, the default is radians (see RAD).

DEL

Keyboard Executable	Yes
Programmable	No

This command deletes program lines.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying a program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
DEL 15
DEL Sort,9999
```

Semantics

DEL cannot be executed while a program is running. If DEL is executed while a program is paused, the computer changes to the stopped state.

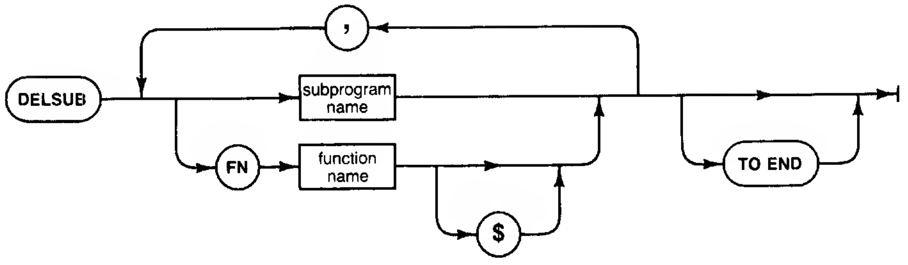
When a line is specified by a line label, the computer uses the lowest numbered line which has the label. If the label does not exist, error 3 is generated. An attempt to delete a non-existent program line is ignored when the line is specified by a line number. An error results if the ending line number is less then the beginning line number. If only one line is specified, only that line is deleted.

When deleting SUB and FN subprograms, the range of lines specified must include the statements delimiting the beginning and ending of the subprogram (DEF FN and FNEND for user-defined function subprograms; SUB and SUBEND for SUB subprograms), as well as all comments following the delimiting statement for the end of the subprogram. Contiguous subprograms may be deleted in one operation.

DELSUB

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deletes one or more SUB subprograms or user-defined function subprograms from memory.



Item	Description/Default	Range Restrictions
subprogram name	name of a SUB subprogram	any valid name
function name	name of a user-defined function	any valid name

Example Statements

```
DELSUB FNTrim$
DELSUB Special1,Special3
```

Semantics

Subprograms being deleted do not need to be contiguous in memory. The order of the names in the deletion list does not have to agree with the order of the subprograms in memory. If there are subprograms with the same name, the one occurring first (lowest line number) is deleted.

The lines deleted begin with the line delimiting the beginning of the subprogram (SUB or DEF FN) and include the comments following the line delimiting the end of the subprogram (SUB-END or FNEND).

You cannot delete:

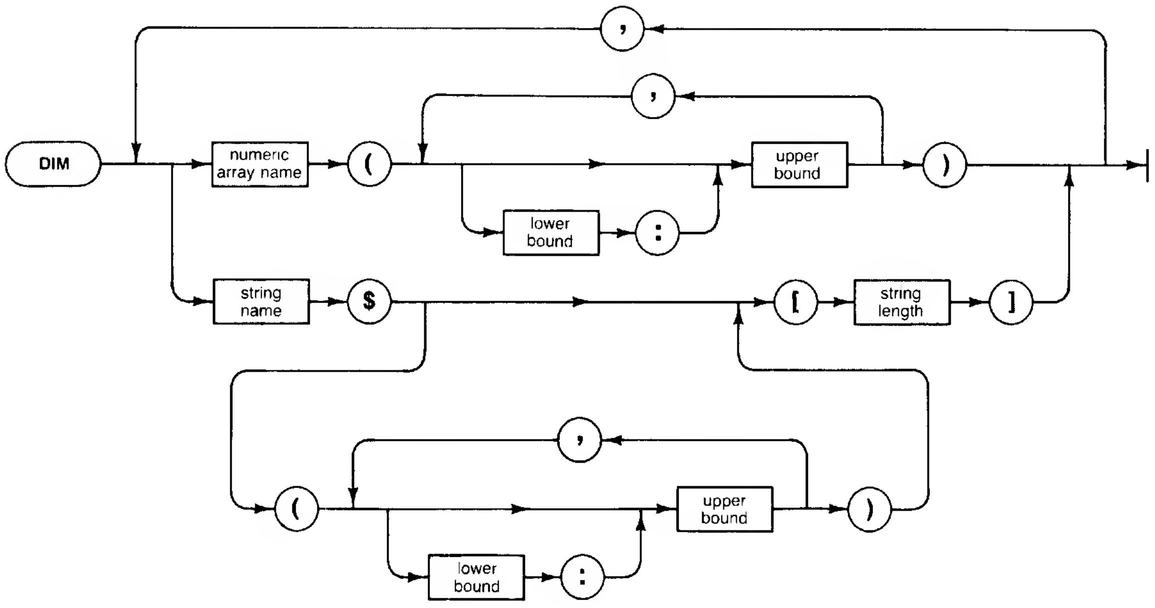
- Busy subprograms (ones being executed).
- Subprograms which are referenced by active ON-event CALL statements.

If an error occurs while attempting to delete a subprogram with a DELSUB statement, the subprogram is not deleted, and neither are subprograms listed to the right of the subprogram which could not be deleted.

DIM

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement dimensions and reserves memory for REAL numeric arrays, strings and string arrays.



Item	Description/Default	Range Restrictions
numeric array name	name of a numeric array	any valid name
string name	name of a string variable	any valid name

lower bound	integer constant;
	Default = OPTION BASE value (0 or 1)
		−32 767 thru +32 767
		(see "array" in Glossary)

upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)
string length	integer constant	1 thru 32 767

Example Statements

```
DIM String$[100],Name$(12)[32]
DIM Array(-128:127,16)
```

Semantics

A program can have any number of DIM statements. The same variable cannot be declared twice within a program (variables declared in a subprogram are distinct from those declared in a main program, except those declared in COM). The DIM statements can appear anywhere within a program, as long as they do not precede an OPTION BASE statement. Dimensioning occurs at pre-run or subprogram entry time. Dynamic run time allocation of memory is provided by the ALLOCATE statement.

No array can have more than six dimensions. Each dimension can have a maximum of 32 767 elements. The actual maximum number of elements for an array depends on available memory.

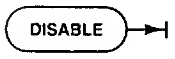
All numeric arrays declared in a DIM statement are REAL, and each element of type REAL requires 8 bytes of storage. A string requires one byte of storage per character, plus two bytes of overhead.

An undeclared array is given as many dimensions as it has subscripts in its lowest-numbered occurrence. Each dimension of an undeclared array has an upper bound of ten. Space for these elements is reserved whether you use them or not.

DISABLE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables event-initiated branches which were defined by ON KEY, ON KNOB, and ON INTR statements.



Sematics

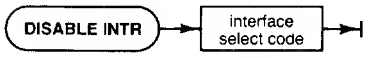
If an event occurs while the event-initiated branches are disabled, the fact that an event has occurred is logged. Although there is an event log for each of the ON-event statements involved, it only records the fact that an event has occurred, there is no record of how many of each type of event has occurred.

If event-initiated branches are enabled after they have been disabled, any ON-event branches for which an event is logged are taken if the system priority permits.

DISABLE INTR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

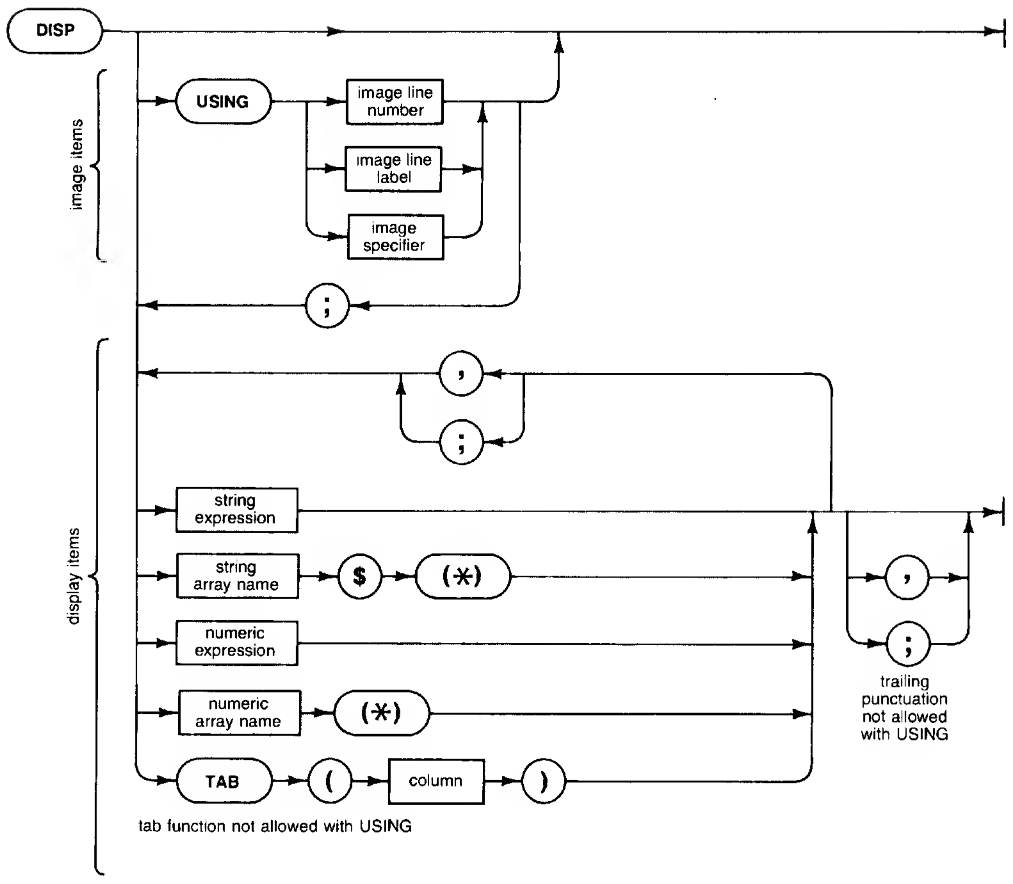
This statement disables interrupts from an interface by turning off the interrupt generating mechanism on the interface.



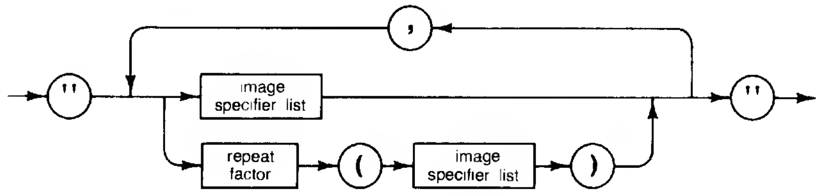
Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
DISABLE INTR 7
DISABLE INTR Isc
```



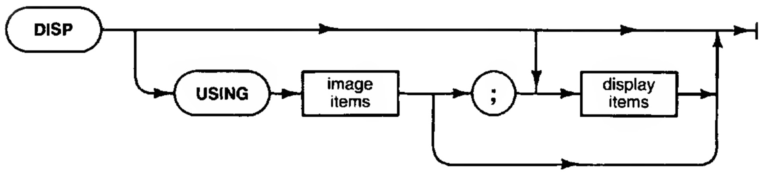
literal form of image specifier:



DISP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement causes the display items to be sent to the display line on the CRT.

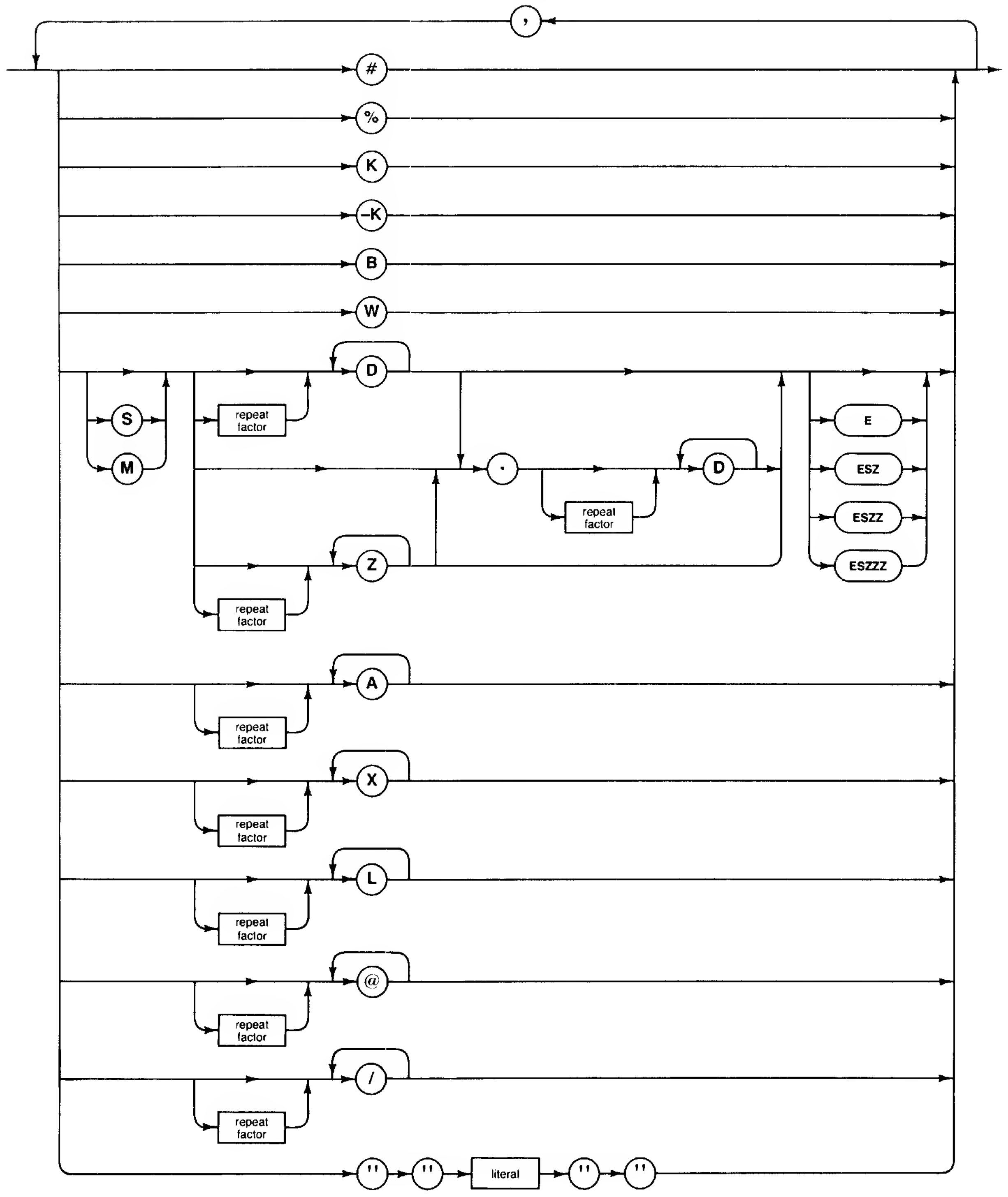


Item	Description/Default	Range Restrictions	Recommended Range
image line label	name identifying an IMAGE statement	any valid name	—
image line number	integer constant identifying an IMAGE statement	1 thru 32 766	—
image specifier	string expression	(see drawing)	—
string array name	name of a string array	any valid name	—
numeric array name	name of a numeric array	any valid name	—
column	numeric expression, rounded to an integer	−32 768 thru +32 767	1 thru 50
image specifier list	literal	(see next drawing)	—
repeat factor	integer constant	1 thru 32 767	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed	—

Example Statements

```
DISP Prompt$;
DISP TAB(5),First,TAB(20),Second
DISP USING "5Z.DD";Money
```

image specifier list



Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E−4 and less than 1E+6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the display list is exhausted, an End Of Line (EOL) sequence is sent to the display line, unless it is suppressed by trailing punctuation or a pound-sign (#) image specifier.

Control Codes

Some ASCII control codes have a special effect in DISP statements:

Character	Keystroke	Name	Action
CHR$(7)	CTRL-G	bell	Sound the beeper
CHR$(8)	CTRL-H	backspace	Move the cursor back one character.
CHR$(12)	CTRL-L	formfeed	Clear the display line.
CHR$(13)	CTRL-M	carriage return	Move cursor to column 1. The next character sent to the display clears the display line, unless it is a carriage return.

Arrays

Arrays may be displayed in their entirety by using the asterisk specifier. They are displayed in row-major order (right-most subscript varies most rapidly) and their format depends on the print mode selected.

Display Without Using

If DISP is used without USING, the punctuation following an item determines the width of the item's display field; a semicolon selects the compact field, and a comma selects the default display field. When the display item is an array with the asterisk array specifier, each array element is considered a separate display item. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the display field to be used for the display item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are displayed with one trailing blank. String items are displayed with no leading or trailing blanks.

The default display field displays items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is displayed with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

In the TAB function, a column parameter less than one is treated as one. A column parameter greater than fifty is treated as fifty.

Display With Using

When the computer executes a DISP USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the display items, the field specifier is acted upon without accessing the display list. When the field specifer requires characters, it accesses the next item in the display list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching display item. If the image specifiers are exhausted before the display items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

The effect of the image specifiers on a DISP statement is shown below:

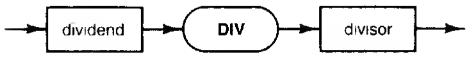
Image Specifier	Meaning
K −K	Compact field. Displays a number or string in standard form with no leading or trailing blanks.
S	Displays the number's sign (+ or −).
M	Displays the number's sign if negative, a blank if positive.
D	Displays one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is displayed, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are displayed.
B	Displays the character represented by one byte of data. This is similar to the CHR$ function. The least significant eight bits of the number are sent. The number is rounded to an integer. If the number is greater than 32 767, 255 is used; if the number is less than −32 768, 0 is used.
W	Displays two characters represented by the two bytes in a 16 bit word. The number is rounded to an integer. If the number is larger than 32 767, 32 767 is used; if the number is less than −32 768, then −32 768 is used. The most significant byte is displayed first, followed by the least significant byte.

Image Specifier	Meaning
A	Displays a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Displays a blank.
.	Displays a decimal point radix indicator.
E ESZZ	Displays an E, a sign, and a two digit exponent.
ESZ	Displays an E, a sign, and a one digit exponent.
ESZZZ	Displays an E, a sign, and a three digit exponent.
#	Suppresses the automatic output of the EOL (End-Of-Line) sequence at the end of the display list.
%	Ignored for display lists.
L	Sends an EOL sequence to the display line.
@	Sends a form-feed to the display line.
/	Sends a carriage-return and a line-feed to the display line.
literal	Displays the characters contained in the literal.

DIV

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns the integer portion of the quotient of the dividend and the divisor.



Item	Description/Default	Range Restrictions
dividend	numeric expression	—
divisor	numeric expression	not equal to 0

Example Statements

```
Quotient=Dividend DIV Divisor
PRINT "Hours =";Minutes DIV 60
```

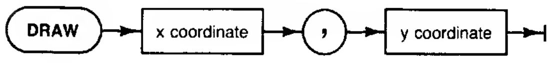
Semantics

DIV returns a REAL value unless both arguments are INTEGER. In the latter case the returned value is INTEGER. A DIV B is identical to SGN(A/B) × INT(ABS(A/B)).

DRAW

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a line from the pen's current position to the specified X and Y coordinate position using the current line type and pen number.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression, in current units	—
y coordinate	numeric expression, in current units	—

Example Statements

```
DRAW 10,90
DRAW Next_x,Next_y
```

Semantics

The X and Y coordinate information is interpreted according to the current unit-of-measure.

The line is clipped at the current clipping boundary. The PIVOT statement rotates the coordinates for the DRAW, but the logical pen position receives the value of the unpivoted coordinates. The logical pen may bear no obvious relationship to the physical pen's position.

A DRAW to the current position generates a point. DRAW updates the logical pen position at the completion of the DRAW statement, and leaves the pen down on an external plotter.

If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

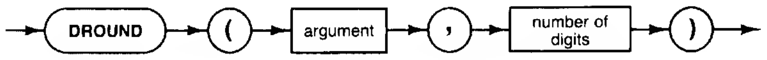
	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

DROUND

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function rounds a numeric expression to the specified number of digits. If the specified number of digits is greater than 15, no rounding takes place. If the number of digits specified is less than 1, 0 is returned.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression	—	—
number of digits	numeric expression, rounded to an integer	—	1 thru 15

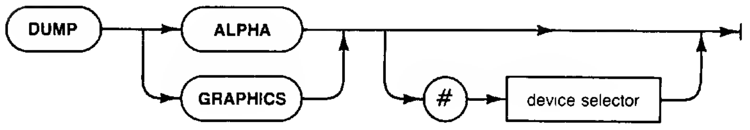
Example Statements

```
Test_real=DROUND(True_real,12)
PRINT "Approx. Volts =";DROUND(Volts,3)
```

DUMP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement copies the contents of the alphanumeric or graphics display to the specified printing device.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer; Default = DUMP DEVICE IS device	external interfaces only (see Glossary)

Example Statements

```
DUMP ALPHA
DUMP GRAPHICS #702
```

Semantics

DUMP ALPHA copies the contents of the CRT alphanumeric display to the specified printer.

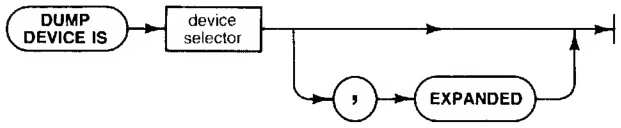
DUMP GRAPHICS copies the contents of the CRT graphics display to a printer. Doing a DUMP GRAPHICS to a printer which does not support the HP Raster Interface Standard will produce unpredictable results. The HP 9876A and the HP 2631G are among the devices which support the standard.

If a DUMP GRAPHICS operation is stopped by pressing the CLR I/O key, the printer may or may not terminate its graphics mode. Sending the printer 75 null characters [CHR$(0)] can be used to terminate the graphics mode on a printer such as the HP 9876.

DUMP DEVICE IS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies which device receives the data when either DUMP ALPHA or DUMP GRAPHICS is executed without a device selector.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer; Default = 701	external interfaces only (see Glossary)

Example Statements

```
DUMP DEVICE IS 721
DUMP DEVICE IS Printer,EXPANDED
```

Semantics

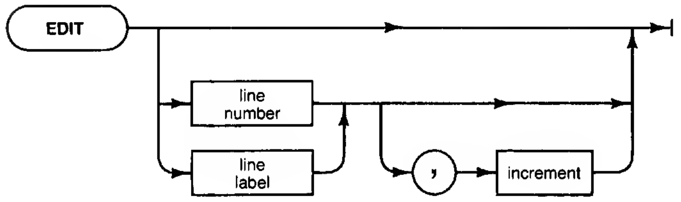
Doing a DUMP GRAPHICS to a printer which does not support the HP Raster Interface Standard will produce unpredictable results. The HP 9876 and the HP 2631G are among the devices which support the standard.

Specifying EXPANDED results in graphics dumps that are twice as big (on each axis) and turned sideways. This gives four dots on the printer for each dot on the display. The resulting picture does not fit on one page of a 9876 or 2631G printer.

EDIT

Keyboard Executable	Yes
Programmable	No

This command allows you to enter a new program or edit a program already in memory.



Item	Description/Default	Range Restrictions
line number	integer constant identifying a program line; Default (see text)	1 thru 32 766
line label	name of a program line	any valid name
increment	integer constant; Default = 10	1 thru 32 766

Example Statements

```
EDIT
EDIT 1000,5
```

Semantics

The EDIT command allows you to scroll through a program in the computer using the arrow keys or knob. Lines may be added to the end of the program by scrolling to the bottom of the program. A new line number will be provided automatically. Lines may be added between existing lines by using the INS LN key. Lines may be deleted using the DEL LN key. Lines may be modifed by typing over the current contents of the line. The ENTER key is used to store newly created or modified lines.

The editor is exited by pressing CONTINUE, CLR SCR, PAUSE, RESET, RUN, or STEP. If the program was changed while paused, pressing CONTINUE generates an error, since modifying a program moves it to the STOP state.

EDIT Without Parameters

If no program is currently in the computer, the edit mode is entered at line 10, and the line numbers are incremented by 10 as each new line is stored. If a program is in the computer, the line number at which the editor enters the program is dependent upon recent history. If an error has paused program execution, the editor enters the program at the line in which the error occurred. Otherwise, the editor enters the program at the line most recently edited (or the beginning of the program after a LOAD or GET operation). The line increment defaults to 10 if it is not specified.

EDIT With Parameters

If no program is in the computer, a number (not a label) must be used to specify the beginning line for the program. The increment will determine the interval between line numbers. If a program is already in the computer, any increment provided is not used until lines are added past the end of the existing program. If a line number is specified between two existing lines, the lowest numbered line greater than the specified line is used. If a line label is used to specify the entry point, the lowest numbered line having that label is used. If the label cannot be found, an error is generated.

ELSE

See the IF...THEN statement.

ENABLE

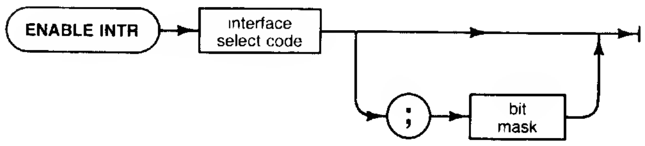
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement re-enables all ON KEY, ON KNOB, and ON INTR branches which were suspended by DISABLE.

ENABLE INTR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement enables the specified interface to generate an interrupt which can cause end-of-statement branches.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
bit mask	numeric expression, rounded to an integer	−32 768 thru +32 767

Example Statements

```
ENABLE INTR 7
ENABLE INTR Isc;Mask
```

Semantics

If a bit mask is specified, its value is stored in the interface's interrupt-enable register. Consult the documentation provided with each interface for the correct interpretation of its bit mask values.

If no bit mask is specified, the previous bit mask for the select code is restored. A bit mask of all zeros is used when there is no previous bit mask.

END

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement marks the end of the main program. (For information about END as a secondary keyword, see the OUTPUT and SEND statements.)

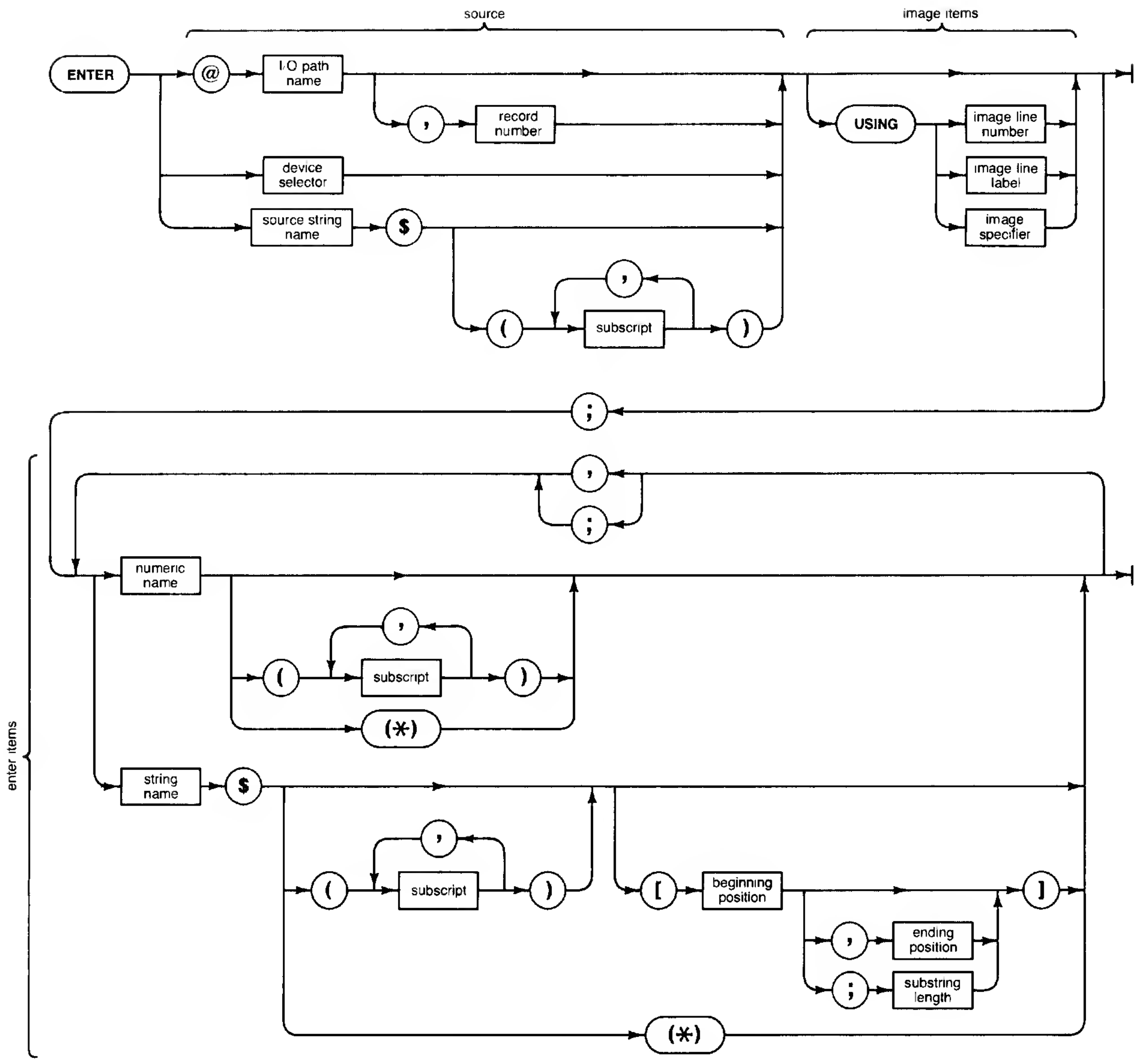
Semantics

END must be the last statement (other than comments) of a main program. Only one END statement is allowed in a program. (Program execution may also be terminated with a STOP statement, and multiple STOP statements are allowed.) END terminates program execution, stops any event-initiated branches, and clears any unserviced event-initiated branches. CONTINUE is not allowed after an END statement.

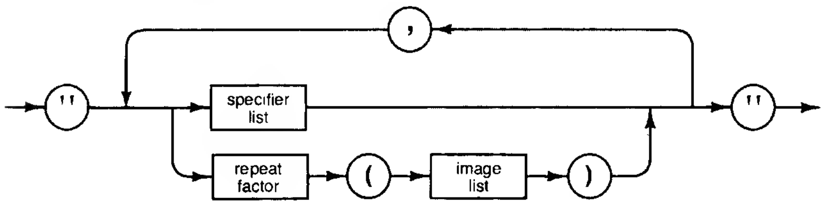
Subroutines used by the main program must occur prior to the END statement. Subprograms and user-defined functions must occur after the END statement.

END IF

See the IF...THEN statement.



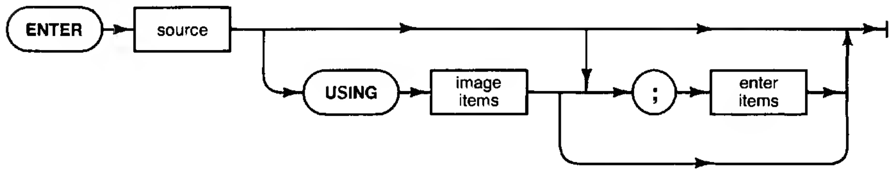
literal form of image specifier:



ENTER

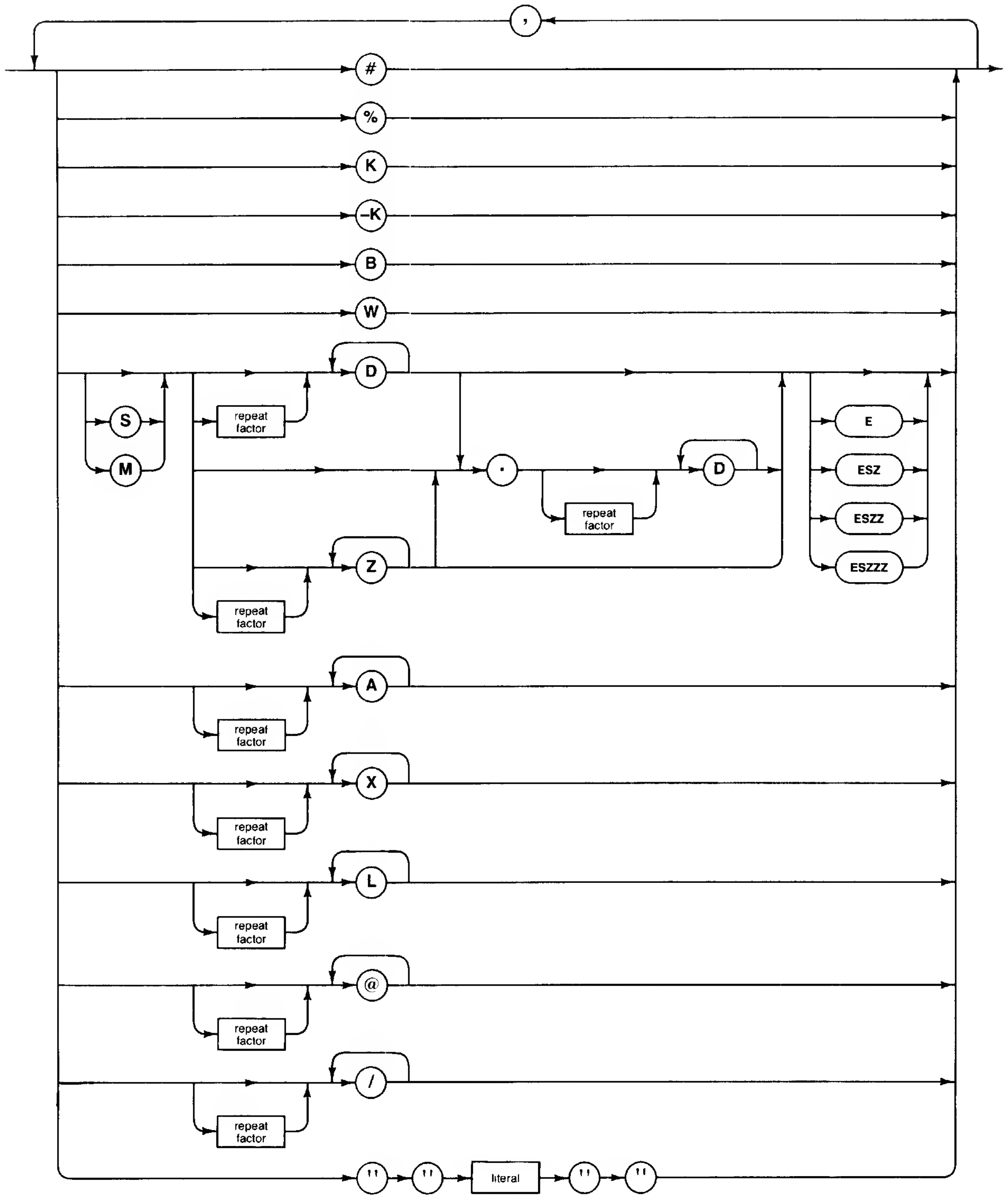
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to input data from a device, file, or string and assign the values entered to variables.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
record number	numeric expression, rounded to an integer	1 thru $2^{31}-1$
device selector	numeric expression, rounded to an integer	(see Glossary)
source string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
image line number	integer constant identifying an IMAGE statement	1 thru 32 766
image line label	name identifying an IMAGE statement	any valid name
image specifier	string expression	(see drawing)
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

image specifier list



Example Statements

```
ENTER 705;Number,String$
ENTER @File;Array(*)
ENTER @Source USING Fmt5;Item(1),Item(2),Item(3)
ENTER 12 USING "#,6A";A$[2;6]
```

Semantics

The Number Builder

If the data being received is ASCII and the associated variable is numeric, a number builder is used to create a numeric quantity from the ASCII representation. The number builder ignores all leading non-numeric characters, ignores all blanks, and terminates on the first non-numeric character, or the first character received with EOI true. (Numeric characters are 0 thru 9, +, −, decimal point, e, and E, in a meaningful numeric order.) If the number cannot be converted to the type of the associated variable, an error is generated. If more digits are received than can be stored in a variable of type REAL, the rightmost digits are lost but any exponent will be built correctly. Overflow occurs only if the exponent overflows.

Arrays

Entire arrays may be entered by using the asterisk specifier. Each element in an array is treated as an item by the ENTER statement, as if the elements were listed separately. The array is filled in row major order (rightmost subscript varies fastest.)

Files as Source

If an I/O path has been assigned to a file, the file may be read with ENTER statements. The file must be an ASCII or BDAT file. The attributes specified in the ASSIGN statement are used only if the file is a BDAT file. Data read from an ASCII file is always in ASCII format. Data read from a BDAT file is considered to be in internal format if FORMAT is OFF, and is read as ASCII characters if FORMAT is ON.

Serial access is available for both ASCII and BDAT files. Random access is available for BDAT files. The file pointer is important to both serial and random access. The file pointer is set to the beginning of the file when the file is opened by an ASSIGN. The file pointer always points to the next byte available for ENTER operations.

Random access uses the record number parameter to read items from a specific location in a file. The record specified must be before the end-of-file. The ENTER begins at the beginning of the specified record.

It is recommended that random and serial access to the same file not be mixed. Also, data should be entered into variables of the same type as those used to output it (e.g. string for string, REAL for REAL, etc.).

Devices as Source

An I/O path name or a device selector may be used to ENTER from a device. If a device selector is used, the default system attributes are used (see ASSIGN). If an I/O path name is used, the ASSIGN statement determines the attributes used. If multiple devices were specified in the ASSIGN, the ENTER sets the first device to be talker, and the rest to be listeners.

If FORMAT ON is the current attribute, the items are read as ASCII. If FORMAT OFF is the current attribute, items are read from the device in the computer's internal format. Two bytes are read for each INTEGER, eight bytes for each REAL. Each string entered consists of a four byte header containing the length of the string, followed by the actual string characters. The string must contain an even number of characters.

CRT as Source
If the device selector is 1, the ENTER is from the CRT. The ENTER reads characters from the CRT, beginnning at the current print position (print position may be modified by using TABXY in a PRINT statement.) The print position is updated as the ENTER progresses. After the last non-blank character in each line, a line-feed is sent with a simulated "EOI". After the eighteenth line is read, the print position is off the screen. If the print position is off screen when an ENTER is started, the off-screen text is first scrolled into line eighteen of the display.

Keyboard as Source
ENTER from device selector 2 may be used to read the keyboard. An entry can be terminated by pressing (ENTER), (CONTINUE), or (STEP). Using (ENTER) or (STEP) causes a CR/LF to be appended to the entry. The (CONTINUE) key adds no characters to the entry and does not terminated the ENTER statement. If an ENTER is stepped into, it is stepped out of, even if the (CONTINUE) key is pressed. An HP-IB EOI may be simulated by pressing (CTRL) (E) before the character to be sent, if this feature has been enabled by an appropriate CONTROL statement to the keyboard (see the Interface Registers in the back of this book).

Strings as Source
If a string name is used as the source, the string is treated similarly to a file. However, there is no file pointer; each ENTER begins at the beginning of the string, and reads serially within the string.

ENTER With USING
When the computer executes an ENTER USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If no variable is required for the field specifier, the field specifier is acted upon without referencing the enter items. When the field specifer references a variable, bytes are entered and used to create a value for the next item in the enter list. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching enter item. If the image specifiers are exhausted before the enter items, the specifiers are reused, starting at the beginning of the specifier list.

Entry into a string variable always terminates when the dimensioned length of the string is reached. If more variables remain in the enter list when this happens, the next character received is associated with the next item in the list.

When USING is specified, all data is interpreted as ASCII characters. FORMAT ON is always assumed with USING, regardless of any attempt to specify FORMAT OFF.

Effects of the image specifiers on an ENTER statement are shown in the following table.

Image Specifier	Meaning
K	Freefield Entry. Numeric: Entered characters are sent to number builder. Leading non-numeric characters are ignored. All blanks are ignored. Trailing non-numeric characters and characters sent with EOI true are delimiters. Numeric characters include digits, decimal point, +, −, e, and E. String: Entered characters are placed in the string. Carriage-return not immediately followed by line-feed is entered into the string. Entry to a string terminates on CR/LF, LF, a character received with EOI true, or when the dimensioned length of the string is reached.
−K	Like K except that LF is entered into a string, and thus CR/LF and LF do not terminate the entry.
S	Same action as D.
M	Same action as D.
D	Demands a character. Non-numerics are accepted to fill the character count. Blanks are ignored, other non-numerics are delimiters.
Z	Same action as D.
B	Demands one byte to become a numeric quantity.
W	Demands two bytes. The two bytes are considered to be a sixteen-bit, two's complement integer. The first byte entered on an eight-bit interface is the most significant byte. A byte of a file or string will be skipped, if necessary, to align data read with a word boundary.
A	Demands a string character. Any character received is placed in the string.
X	Skips a character.
.	Same action as D.
E ESZZ	Same action as 4D.
ESZ	Same action as 3D.
ESZZZ	Same action as 5D.
#	Suppresses all statement terminating conditions; enter data until variable list is satisfied.
%	EOI (or end-of-file) is an immediate statement terminator. No statement terminator is required.

Image Specifier	Meaning
/	Demands a new field; skips all characters to the next line-feed. EOI is ignored.
L	Ignored for ENTER.
@	Ignored for ENTER.
literal	Skips one character for each character in the literal.

ENTER Statement Termination

A simple ENTER statement (one without USING) expects to give values to all the variables in the enter list and then receive a statement terminator. A statement terminator is an EOI, a line-feed received at the end of the last variable or within 256 characters after the end of the last variable, or an end-of-file. If a statement terminator is received before all the variables are satisfied, or no terminator is received within 256 bytes after the last variable is satisfied, an error occurs. The terminator requirements can be altered by using images.

An ENTER statement with USING, but without a % or # image specifier, is different from a simple ENTER in one respect. EOI is not treated as a statement terminator unless it occurs on or after the last variable. Thus, EOI is treated like line-feed and can be used to terminate entry into each variable.

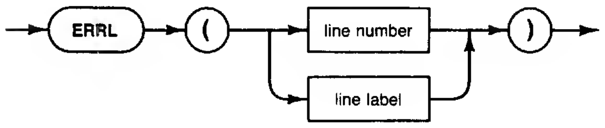
An ENTER statement with USING that specifies a # image requires no terminator. EOI and line-feed serve to end the entry into individual variables. The ENTER statement terminates when the variable list has been satisfied.

An ENTER statement with USING that specifies a % image allows EOI as a statement terminator. Like the # specifier, no terminator is required. Unlike the # specifier, if an EOI is received, it is treated as a statement terminator. If the EOI occurs at a normal boundary between items, the ENTER statement terminates without error and leaves the value of any remaining variables unchanged.

ERRL

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value of 1 if the most recent error occurred in the specified line. Otherwise, a value of 0 is returned. The specified line must be in same context as the ERRL function.



Item	Description/Default	Range Restrictions
line number	integer constant	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
IF ERRL(220) THEN Parse_error
IF NOT ERRL(Parameters) THEN Other
```

ERRN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the number of the most recent program execution error. If no error has occurred, a value of 0 is returned.

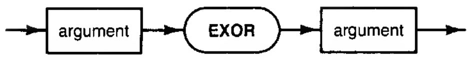
Example Statements

```
IF ERRN=80 THEN Disc_out
DISP "Error Number";ERRN
```

EXOR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based on the logical exclusive-or of its arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

Example Statements

```
Ok=First_pass EXOR Old_data
IF A EXOR Flag THEN Exit
```

Semantics

A non-zero value (positive or negative) is treated as a logical 1; only a zero is treated as a logical 0.

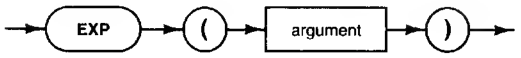
The EXOR function is summarized in this table.

A	B	A EXOR B
0	0	0
0	1	1
1	0	1
1	1	0

EXP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

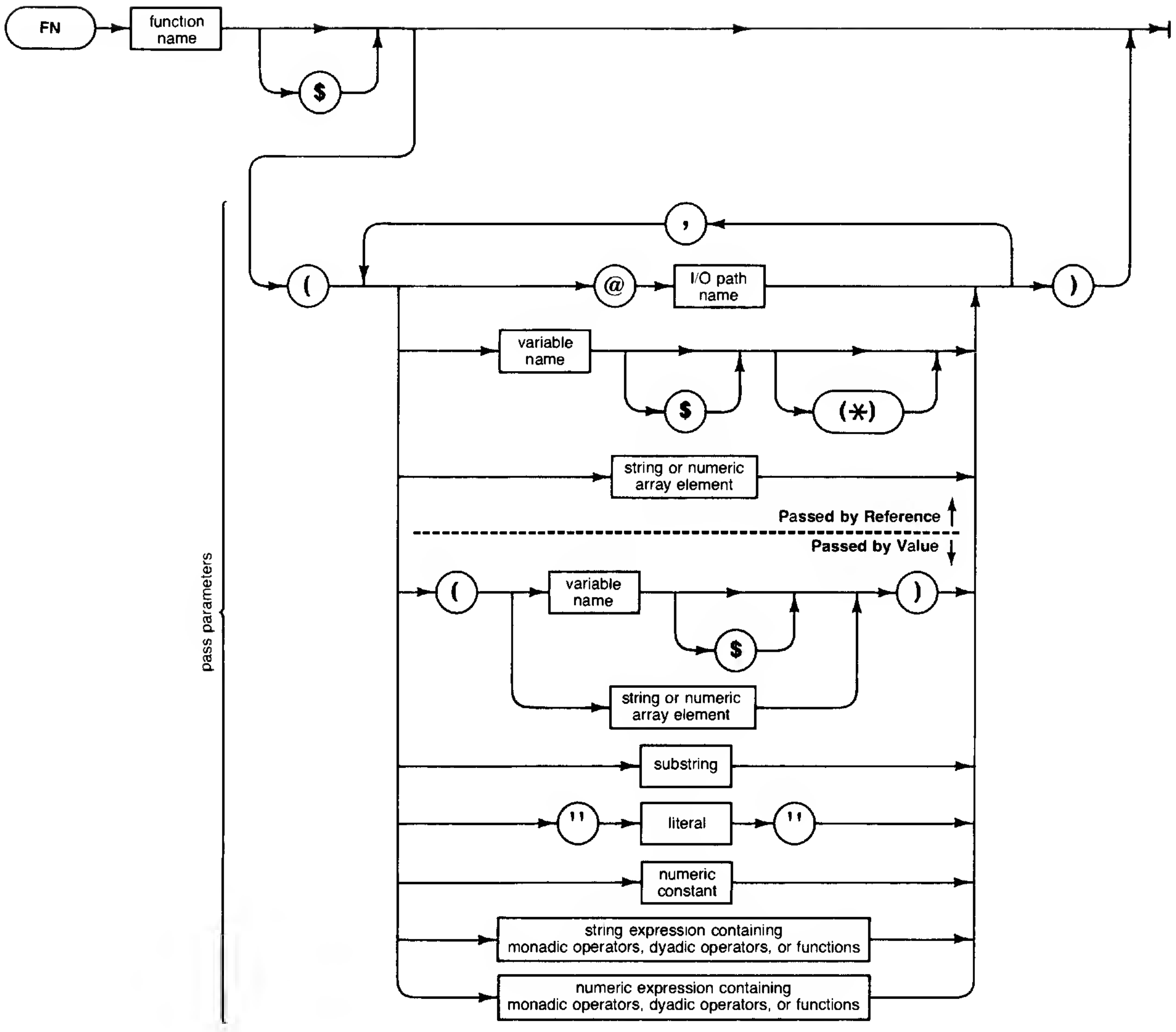
This function raises e to the power of the argument. In the computer, Naperian e ≈ 2.718 281 828 459 05.



Item	Description/Default	Range Restrictions
argument	numeric expression	− 708.396 418 532 264 thru + 709.782 712 893 383 8

Example Statements

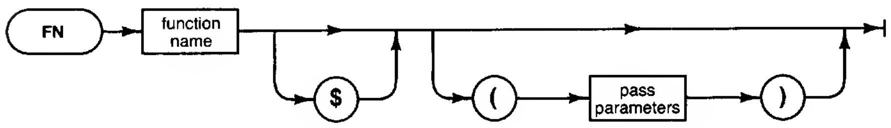
```
Y=EXP(-X^2/2)
PRINT "e to the";Z;"=";EXP(Z)
```


FN
function name
$
,
(
@
I/O path name
)
variable name
$
(*)
string or numeric array element
Passed by Reference
Passed by Value
(
variable name
$
)
string or numeric array element
substring
"
literal
"
numeric constant
string expression containing monadic operators, dyadic operators, or functions
numeric expression containing monadic operators, dyadic operators, or functions
pass parameters

FN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This keyword transfers program execution to the specified user-defined function and may pass items to the function. The value returned by the function is used in place of the function call when evaluating the statement containing the function call.



Item	Description/Default	Range Restrictions
function name	name of a user-defined function	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
variable name	name of a numeric or string variable	any valid name
substring	string expression containing substring notation	(see Glossary)
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	—
numeric constant	numeric quantity expressed using numerals, and optionally a sign, decimal point, or exponent notation	—

Example Statements

```
PRINT X;FNChange(X)
Final$=FNTrim$(First$)
Result=FNPround(Item,Power)
```

Semantics

A user-defined function may be invoked as part of a stored program line or as part of a statement executed from the keyboard. If the function name is typed and then (EXECUTE) is pressed, the value returned by the function is displayed. In order to be invoked from the keyboard, the program containing the function must be running or must have been run. The dollar sign suffix indicates that the returned value will be a string. User-defined functions are created with the DEF FN statement.

The pass parameters must be of the same type (numeric or string) as the corresponding parameters in the DEF FN statement. Numeric values passed by value are converted to the numeric type (REAL or INTEGER) of the corresponding formal parameter. Variables passed by reference must match the type of the corresponding parameter in the DEF FN statement exactly. An entire array may be passed by reference by using the asterisk specifier.

Invoking a user-defined function changes the program context. The functions may be invoked recursively.

If there is more than one user-defined function with the same name, the lowest numbered one is invoked by FN.

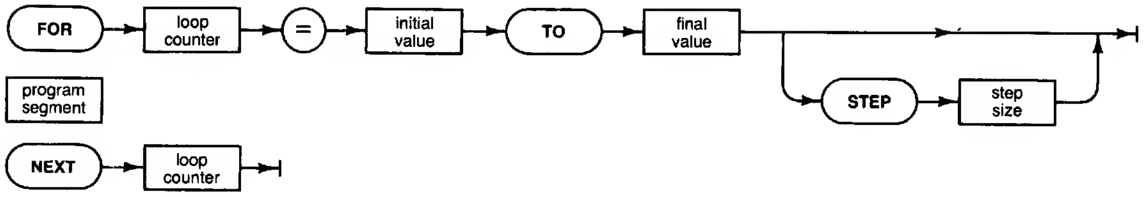
FNEND

See the DEF FN statement.

FOR...NEXT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the loop counter passes a specific value. The step size may be positive or negative.



Item	Description/Default	Range Restrictions
loop counter	name of a numeric variable	any valid name
initial value	numeric expression	—
final value	numeric expression	—
step size	numeric expression; Default = 1	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Program Segments

```
100  FOR I=4 TO 0 STEP -.1
110    PRINT I;SQR(I)
120  NEXT I

1220  INTEGER Point
1230  FOR Point=1 TO LEN(A$)
1240    CALL Convert(A$[Point;1])
1250  NEXT Point
```

Semantics

The loop counter is set equal to the initial value when the loop is entered. Each time the corresponding NEXT statement is encountered, the step size (which defaults to 1) is added to the loop counter, and the new value is tested against the final value. If the final value has not been passed, the loop is executed again, beginning with the line immediately following the FOR statement. If the final value has been passed, program execution continues at the line following the NEXT statement. Note that the loop counter is not equal to the specified final value when the loop is exited.

The loop counter is also tested against the final value as soon as the values are assigned when the loop is first entered. If the loop counter has already passed the final value in the direction the step would be going, the loop is not executed at all. The loop may be exited arbitrarily (such as with a GOTO), in which case the loop counter has whatever value it had obtained at the time the loop was exited.

Each FOR statement is allowed one and only one matching NEXT statement. The NEXT statement must be in the same context as the FOR statement. FOR...NEXT loops may be nested, and may be contained in IF...THEN...ELSE constructs, as long as the loops and constructs are properly nested and do not improperly overlap.

The initial, final and step size values are calculated when the loop is entered and are used while the loop is repeating. If a variable or expression is used for any of these values, its value may be changed after entering the loop without affecting how many times the loop is repeated. However, changing the value of the loop counter itself can affect how many times the loop is repeated.

The loop counter variable is allowed in expressions that determine the initial, final, or step size values. The previous value of the loop counter is not changed until after the initial, final, and step size values are calculated.

If the step value evaluates to 0, the loop repeats infinitely and no error is given.

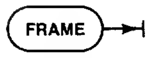
FORMAT

See the ASSIGN statement.

FRAME

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a frame around the current clipping area using the current pen number and line type. After drawing the frame, the current pen position coincides with the lower left corner of the frame, and the pen is down.



GCLEAR

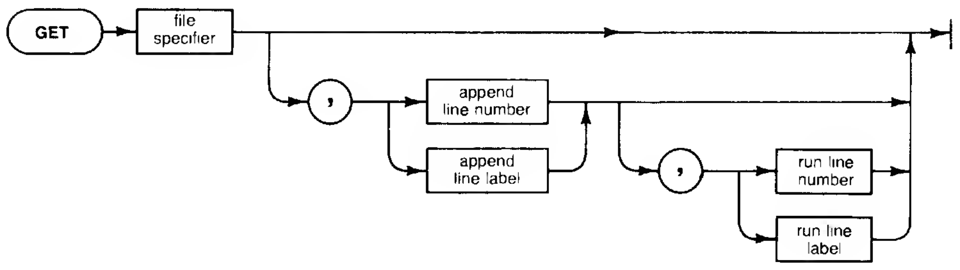
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement clears the graphics display or sends a command to an external plotter to advance the paper.

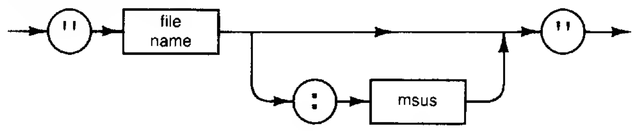
GET

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement reads the specified ASCII file and attempts to store the strings into memory as program lines.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

append line number	integer constant identifying a program line	1 thru 32 766
append line label	name of a program line	any valid name
run line number	integer constant identifying a program line	1 thru 32 766
run line label	name of a program line	any valid name

Example Statements

```
GET "George"
GET Next_prog$,180,10
```

Semantics

When GET is executed, the first line in the specified file is read and checked for a valid line number. If no valid line number is found, the current program stays in memory and error 68 is generated. If the GET was attempted from a running program, the program remains active and the error 68 can be trapped with ON ERROR. If there is no ON ERROR in effect, the program pauses.

If there is a valid line number at the start of the first line in the file, the GET operation proceeds. Values for all variables except those in COM are lost and the current program is deleted from the append line to the end. If no append line is specified, the entire current program is deleted.

As the file is brought in, each line is checked for proper syntax. Any lines which contain syntax errors are listed on the PRINTER IS device. Those erroneous lines which have valid line numbers are converted into comments and stored in the program. The syntax checking during GET is the same as if the lines were being typed from the keyboard, and any errors that would occur during keyboard entry will also occur during GET. If any line caused a syntax error, an error 68 is reported at the completion of the GET operation. This error is not trappable because the old program was deleted and the new one is not running yet.

Any line in the main program or any subprogram may be used for the append location. If an append line number is specified, the lines from the file are renumbered by adding an offset to their line numbers. This offset is the difference between the append line number and the first line number in the file. This operation preserves the line-number intervals that exist in the file. If renumbering would create an invalid line number, the line causing the error is listed on the PRINTER IS device showing the line number it had in the file. Any programmed references to line numbers that would be renumbered by REN are also renumbered by GET. If no append line is specified, the lines from the file are entered without renumbering.

If a successful GET is executed from a program, execution resumes automatically after a prerun initialization (see RUN). If no run line is specified, execution resumes at the lowest-numbered line in the program. If a run line is specified, execution resumes at the specified line. The specified run line must be a line in the main program segment.

If a successful GET is executed from the keyboard **and** a run line is specified, a prerun is performed and program execution begins automatically at the specified line. If GET is executed from the keyboard with no run line specified, RUN must be executed to start the program. GET is not allowed from the keyboard while a program is running.

GINIT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement establishes a set of default values for variables affecting graphics operations.

Semantics

The following operations are performed when GINIT is executed:

```
PLOTTER IS 3,"INTERNAL"
CLIP OFF
PIVOT 0
PEN 1
LINE TYPE 1,5
LORG 1
CSIZE 5,0.6
LDIR 0
MOVE 0,0
```

In addition, if the next graphics statement encountered is **not** a PLOTTER IS with an HPGL specifier, the CRT graphics raster is cleared, and the following statements are executed:

```
VIEWPORT 0,133.444816054,0,100
WINDOW 0,133.444816054,0,100
```

If the next graphics statement is a PLOTTER IS with an HPGL plotter specifier, and the plotter has a horizontal aspect ratio (X-axis longer than the Y-axis) the following statements are executed:

```
VIEWPORT 0,RATIO*100,0,100
WINDOW 0,RATIO*100,0,100
```

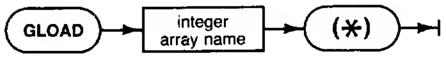
If the next graphics statement is a PLOTTER IS with an HPGL plotter specifier, and the plotter has a vertical aspect ratio (Y-axis longer than the X-axis) the following statements are executed:

```
VIEWPORT 0,100,0,RATIO*100
WINDOW 0,100,0,RATIO*100
```

GLOAD

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement loads the contents of an INTEGER array into the graphics display memory (also see GSTORE).



Item	Description/Default	Range Restrictions
integer array name	name of an INTEGER array with exactly 7500 elements	any valid name

Example Statements

```
GLOAD Picture(*)
IF Flag THEN GLOAD Array(*)
```

Semantics

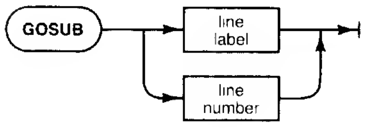
A binary one represents a pixel that is turned on, while a binary zero represents a pixel that is turned off. A pixel is the smallest point that can be independently turned on and off on a CRT. The graphics display on the 9826 is 300 by 400 pixels.

The upper left corner of the CRT is represented by the most significant bit of the lowest numbered element in the array. Each array element represents 16 pixels on the CRT. A full row of dots is contained in 25 sequential array elements (16 × 25 = 400 dots).

GOSUB

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to the subroutine at the specified line. The specified line must be in the current context. The current program line is remembered in anticipation of returning (see RETURN).



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766

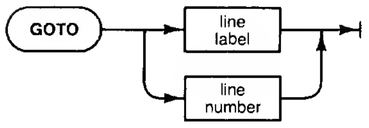
Example Statements

```
GOSUB 120
IF Numbers THEN GOSUB Process
```

GOTO

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to the specified line. The specified line must be in the current context.



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766

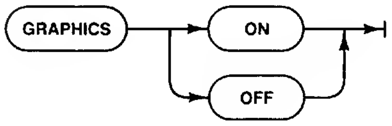
Example Statements

```
GOTO 550
GOTO Loop_start
IF Full THEN Exit
```

GRAPHICS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement turns the graphics display on or off. This statement has no effect on the contents of the graphics memory, it just controls whether it is displayed or not. At power-on, after RESET, or after SCRATCH A, the graphics display is off.



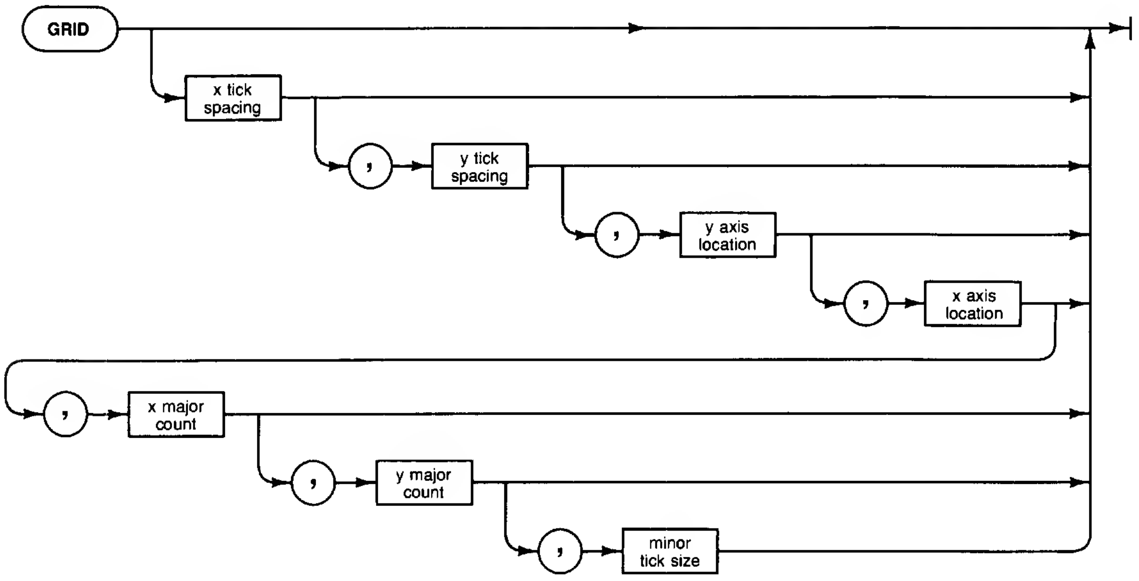
Example Statements

```
GRAPHICS ON
IF Flag THEN GRAPHICS OFF
```

GRID

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a full grid pattern. The pen is left at the intersection of the X and Y axes.



Item	Description/Default	Range Restrictions
x tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y tick spacing	numeric expression in current units; Default = 0, no ticks	(see text)
y axis location	numeric expression specifying the location of the y axis in x-axis units; Default = 0	—
x axis location	numeric expression specifying the location of the x axis in y-axis units; Default = 0	—
x major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
y major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks; Default = 1 (every tick is major)	1 thru 32 767
minor tick size	numeric expression in graphic display units; Default = 2	—

Example Statements

```
GRID 10,10,0,0
GRID Xmin,Ymin,Xintercept,Yintercept,5,5
```

Semantics

Grids are drawn with the current line type and pen number. Major tick marks are drawn as lines across the entire soft clipping area. A cross tick is drawn at the intersection of minor tick marks.

The X and Y tick spacing must not generate more than 32 768 grid marks in the clip area, or error 20 will be generated. To insure generation of a complete grid, the X and Y axis locations must both lie within the current clip area.

Applicable Graphics Transformations

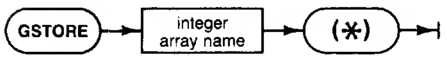
	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

GSTORE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement stores the contents of the graphics display memory into an INTEGER array (also see GLOAD).



Item	Description/Default	Range Restrictions
integer array name	name of an INTEGER array with exactly 7500 elements	any valid name

Example Statements

```
GSTORE Picture(*)
IF Final THEN GSTORE A(*)
```

Semantics

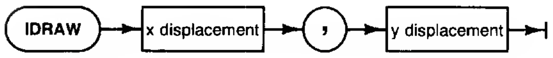
A binary one represents a pixel that is turned on, while a binary zero represents a pixel that is turned off. A pixel is the smallest point that can be independently turned on and off on a CRT. The graphics display on the 9826 is 300 by 400 pixels.

The upper left corner of the CRT is represented by the most significant bit of the lowest numbered element in the array. Each array element represents 16 pixels on the CRT. A full row of dots is contained in 25 sequential array elements (16 × 25 = 400 dots).

IDRAW

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement draws a line from the current pen position to a position calculated by adding the X and Y displacements to the current pen position.



Item	Description/Default	Range Restrictions
x displacement	numeric expression in current units	—
y displacement	numeric expression in current units	—

Example Statements

```
IDRAW X+50,0
IDRAW Delta_x,Delta_y
```

Semantics

The X and Y displacement information is interpreted according to the current unit-of-measure.

The line is clipped at the current clipping boundary. The PIVOT statement rotates the coordinates for the IDRAW, but the logical pen position receives the value of the unpivoted coordinates. The logical pen may bear no obvious relationship to the physical pen's position.

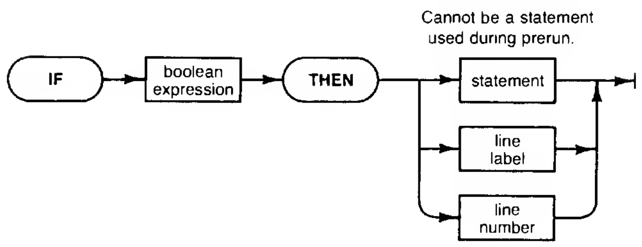
An `IDRAW 0,0` generates a point. IDRAW updates the logical pen position at the completion of the IDRAW statement, and leaves the pen down on an external plotter.

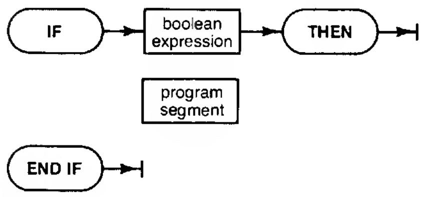
If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

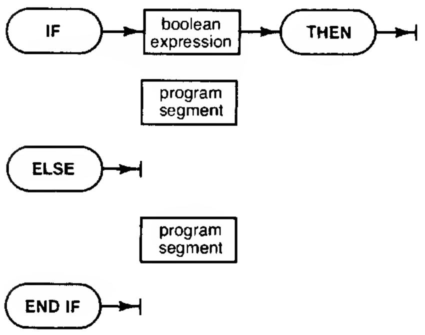
IF...THEN

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement provides conditional branching.







Item	Description/Default	Range Restrictions
boolean expresion	numeric expression; evaluated as true if non-zero and false if zero	—
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
statement	a programmable statement	(see following list)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Program Segments

```
150   IF Flag THEN Next_file
160   IF Pointer<1 THEN Pointer=1

580   IF First_pass THEN
590     Flag=0
600     INPUT "Command?",Cmd$
610     IF LEN(Cmd$) THEN GOSUB Parse
620   END IF

1000   IF X<0 THEN
1010     BEEP
1020     DISP "Improper Argument"
1030   ELSE
1040     Root=SQR(X)
1050   END IF
```

Semantics

If the boolean expression evaluates to 0, it is considered false; if the evaluation is non-zero, it is considered true. Note that a boolean expression can be constructed with numeric or string expressions separated by relational operators, as well as with a numeric expression.

Single Line IF...THEN

If the conditional statement is a GOTO, execution is transferred to the specified line. The specified line must exist in the current context. A line number or line label by itself is considered an implied GOTO. For any other statement; the statement is executed, then program execution resumes at the line following the IF...THEN statement. If the tested condition is false, program execution resumes at the line following the IF...THEN statement, and the conditional statement is not executed.

Prohibited Statements

The following statements must be identified at prerun time or are not executed during normal program flow. Therefore, they are not allowed as the statement in a single line IF...THEN construct.

COM	END	IMAGE	REM
DATA	END IF	INTEGER	SUB
DEF FN	FNEND	NEXT	SUBEND
DIM	FOR	OPTION BASE	
ELSE	IF	REAL	

Multiple Line IF...THEN...ELSE...END IF

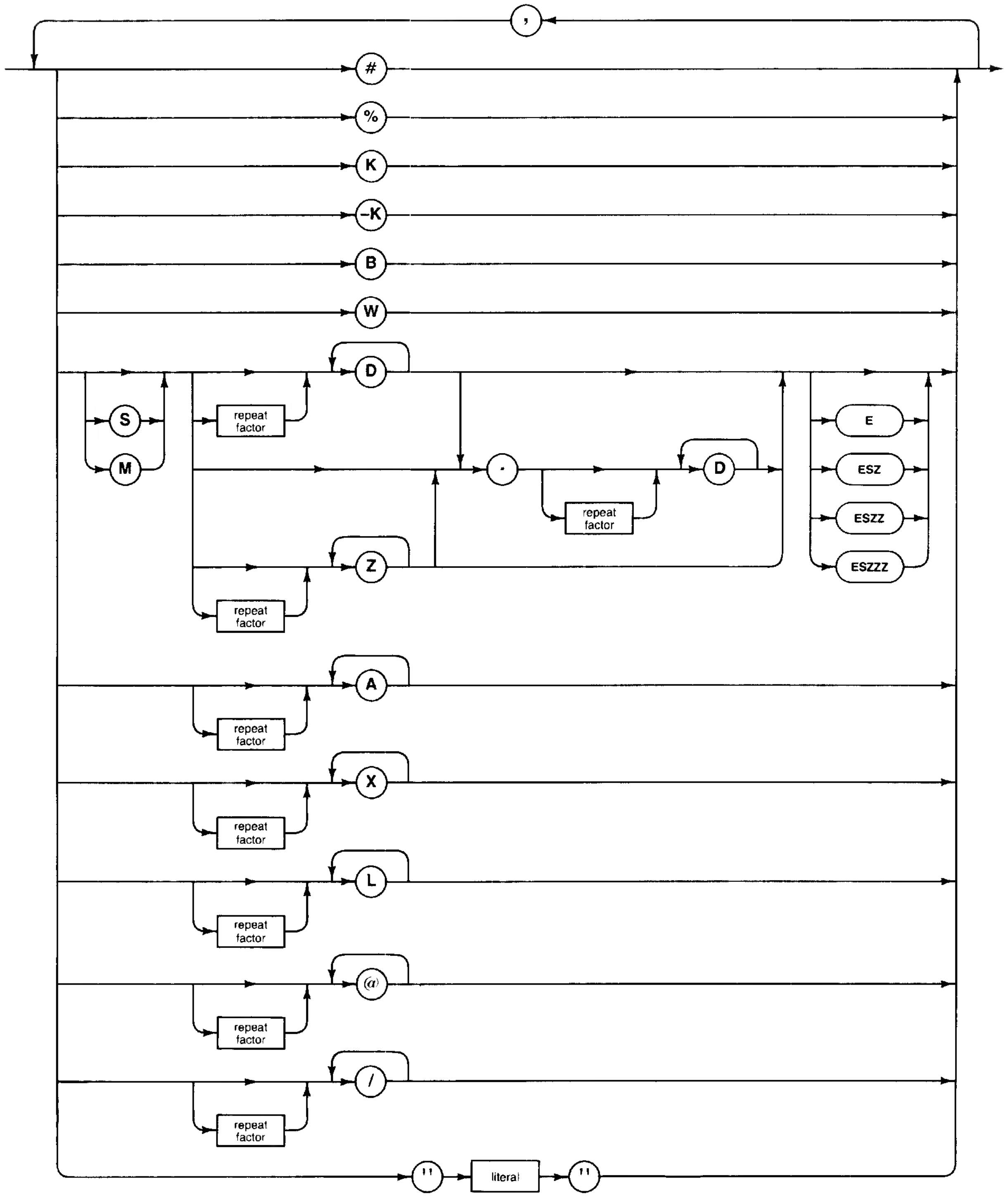
The IF...THEN...END IF construct allows a multiple-statement program segment to be specified between the IF and the END IF. This conditional program segment is executed if the numeric expression is true (non-zero). Program execution continues with the statement after the END IF if the expression is false (zero).

When ELSE is specified, only one of the program segments will be executed. When the condition is true, the segment between IF...THEN and ELSE is executed. When the condition is false, the segment between ELSE and END IF is executed. In either case, when the construct is exited, program execution continues with the statement after the END IF.

Branching into an IF...THEN construct (such as with a GOTO) results in a branch to the program line following the END IF when the ELSE statement is executed.

The prohibited statements listed above are allowed in multiple-line IF...THEN constructs. However, these statements are not executed conditionally. The exceptions are FOR...NEXT loops and other IF...THEN statements or constructs. These are executed conditionally, but need to be properly nested. To be properly nested, the entire FOR...NEXT loop or IF...THEN construct must be contained in one program segment (see drawing). Similarly, when IF...THEN constructs are used in FOR...NEXT loops, they must be entirely contained in the loop.

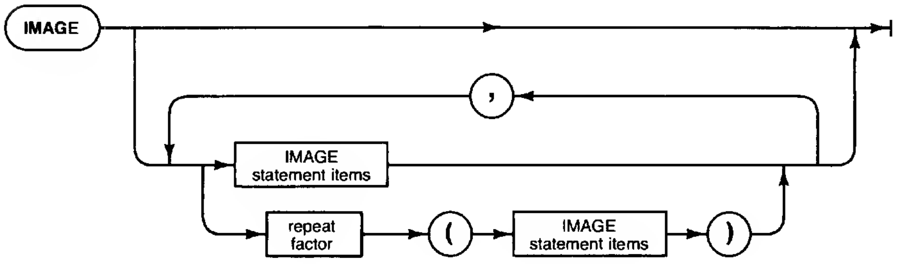
IMAGE statement items



IMAGE

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement provides image specifiers for the ENTER, OUTPUT, DISP, LABEL, and PRINT statements. Refer to the appropriate statement for details on the effect of the various image specifiers.



Item	Description/Default	Range Restrictions
IMAGE statement items	literal	(see drawing)
repeat factor	integer constant	1 thru 32 767
literal	string composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

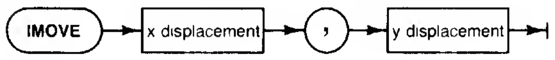
Example Statements

```
IMAGE 4Z.DD,3X,K,/
IMAGE "Result = ",SDDDE,3(XX,ZZ)
IMAGE #,B
```

IMOVE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement updates the logical pen position, by adding the X and Y displacements to the current logical pen position. The physical pen is not moved until an operation is performed which requires the pen to draw something. The logical pen may bear no obvious relation to the physical pen. The X and Y increments are interpreted according to the current unit-of-measure.



Item	Description/Default	Range Restrictions
x displacement	numeric expression in current units	—
y displacement	numeric expression in current units	—

Example Statements

```
IMOVE X+50,0
IMOVE Delta_x,Delta_y
```

Applicable Graphics Transformations

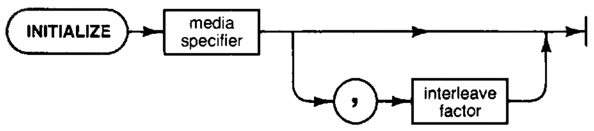
	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

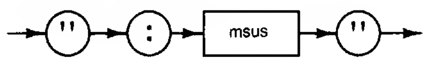
INITIALIZE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement prepares mass storage media for use by the 9826. When INITIALIZE is executed, **any data on the media is lost**.



literal form of media specifier:



Item	Description/Default	Range Restrictions	Recommended Range
media specifier	string expression	(see drawing)	—
msus	literal	INTERNAL	—
interleave factor	numeric expression, rounded to an integer and evaluated MOD 16; 0 is treated as 1; Default = 1	−32 768 thru +32 767	1 thru 15

Example Statements

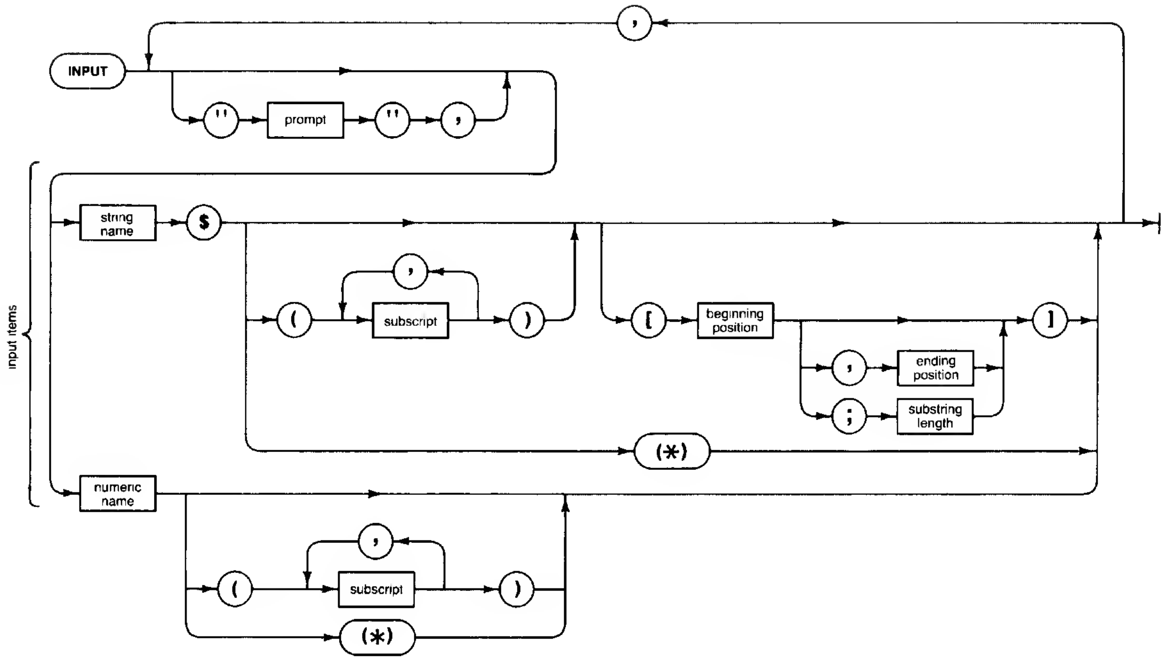
```
INITIALIZE ":INTERNAL"
INITIALIZE Disc$,2
```

Semantics

Any media used by the computer must be initialized before its **first** use. Initialization rewrites the directory, eliminating any access to old data. The media is partitioned into 256-byte physical records. The quality of the media is checked during initialization. Defective tracks are "spared" (marked so that they will not be used).

The interleave factor establishes the distance in physical records between consecutively numbered records. If the interleave factor evaluates to 0, 1 is used. The interleave factor is ignored if the mass storage device is not a disc.

The msus for the internal mini-floppy disc drive is INTERNAL.

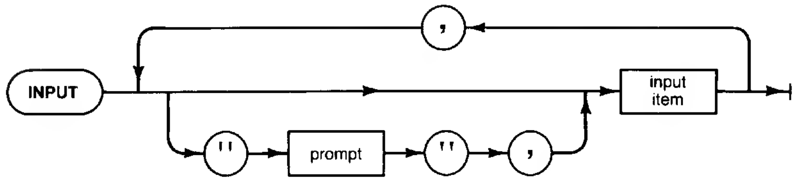


INPUT
"
prompt
"
,
,
input items
string name
$
(
subscript
,
)
[
beginning position
,
ending position
;
substring length
]
(*)
numeric name
(
subscript
,
)
(*)

INPUT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to assign keyboard input to program variables.



Item	Description/Default	Range Restrictions
prompt	a literal composed of characters from the keyboard, including those generated using the ANY CHAR key; Default = question mark	—
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
numeric name	name of a numeric variable	any valid name

Example Statements

```
INPUT "Name?",N$,"ID Number?",Id
INPUT Array(*)
```

Semantics

Values can be assigned through the keyboard for any numeric or string variable, substring, array, or array element.

A prompt, which is allowed for each item in the input list, appears on the CRT display line. If the last DISP or DISP USING statement suppressed its EOL sequence, the prompt is appended to the current display line contents. If the last DISP or DISP USING did not suppress the EOL sequence, the prompt replaces the current display line contents.

Not specifying a prompt results in a question mark being used as the prompt. Specifying the null string (" ") for the prompt suppresses the question mark.

To respond to the prompt, the operator enters a number or a string. Leading and trailing blank characters are deleted. Unquoted strings may not contain commas or quote marks. Placing quotes around an input string allows any characters to be used as input. If " is intended to be a character in a quoted string, use " ".

Multiple values can be entered individually or separated by commas. Press the (CONTINUE), (ENTER) or (STEP) after the final input response. Two consecutive commas cause the corresponding variable to retain its original value. Terminating an input line with a comma retains the old values for all remaining variables in the list.

The assignment of a value to a variable in the INPUT list is done as soon as the terminator (comma or key) is encountered. Not entering data and pressing (CONTINUE), (ENTER), or (STEP) retains the old values for all remaining variables in the list.

If (CONTINUE) or (ENTER) is pressed to end the data input, program execution continues at the next program line. If (STEP) is pressed, the program execution continues at the next program line in single step mode. (If the INPUT was stepped into, it is stepped out of, even if (CONTINUE) or (ENTER) is pressed.)

If too many values are supplied for an INPUT list, the extra values are ignored.

An entire array may be specified by the asterisk specifier. Inputs for the array are accepted in row major (right most subscript varies most rapidly).

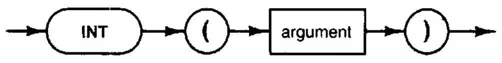
Live keyboard operations are not allowed while an INPUT is awaiting data entry. (PAUSE) can be pressed so live keyboard operations can be performed. The INPUT statement is re-executed, beginning with the first item, when (CONTINUE) or (STEP) is pressed. All values for that particular INPUT statement must be re-entered.

ON KEY and ON KNOB events are deactivated during an INPUT statement. Errors do not cause an ON ERROR branch. If an input response results in an error, re-entry begins with the variable which would have received the erroneous response.

INT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the greatest integer which is less than or equal to the expression. The result will be of the same type (REAL or INTEGER) as the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

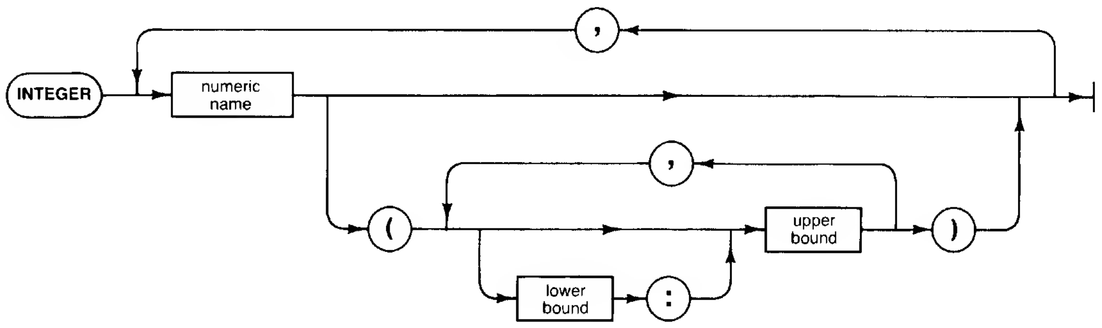
Example Statements

```
Whole=INT(Number)
IF X/2=INT(X/2) THEN Even
```

INTEGER

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement declares INTEGER variables, dimensions INTEGER arrays, and reserves memory for them. (For information about INTEGER as a secondary keyword, see the ALLOCATE, COM, DEF FN, or SUB statements.)



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or 1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)

Example Statements

```
INTEGER I,J,K
INTEGER Array(-128:255)
```

Semantics

An INTEGER variable (or an element of an INTEGER array) uses two bytes of storage space. An INTEGER array can have a maximum of six dimensions. The maximum number of elements is a function of your computer's memory size, but no single dimension can have more than 32 767 total elements.

KNOBX

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the net number of knob pulses counted since the last time the KNOBX counter was zeroed.

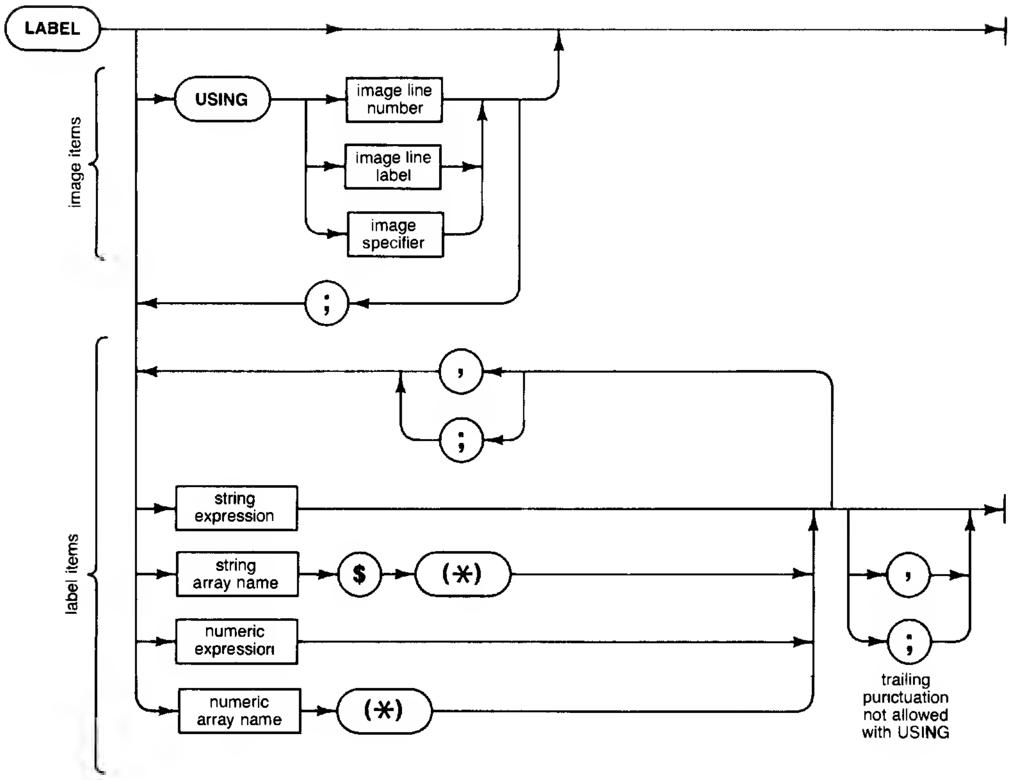
Example Statements

```
Position=KNOBX
IF KNOBX<0 THEN Backwards
```

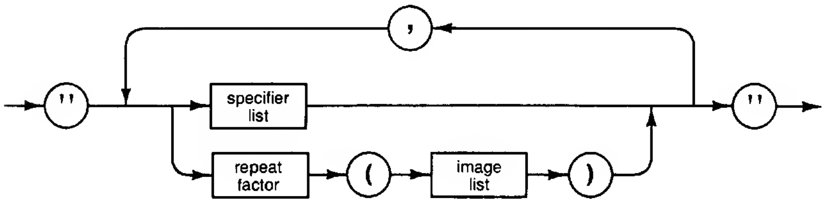
Semantics

Sampling occurs during the time interval established by the ON KNOB statement. The counter is zeroed when the KNOBX function is called and at the times specified in the Reset Table at the back of this manual. Clockwise rotation gives positive counts; counter-clockwise rotation gives negative counts. There are 120 counts for one revolution of the knob. If there is no active ON KNOB definition, KNOBX returns zero.

Counts are accumulated by the KNOBX function during each ON KNOB sampling interval. The pulse count during each sampling interval is limited to −127 thru +128. The limits of the KNOBX function are −32 768 thru +32 767.



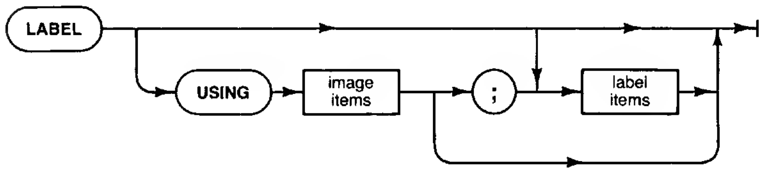
literal form of image specifier:



LABEL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement produces alphanumeric labels on graphic devices. (For information about LABEL as a secondary keyword, see the ON KEY statement.)

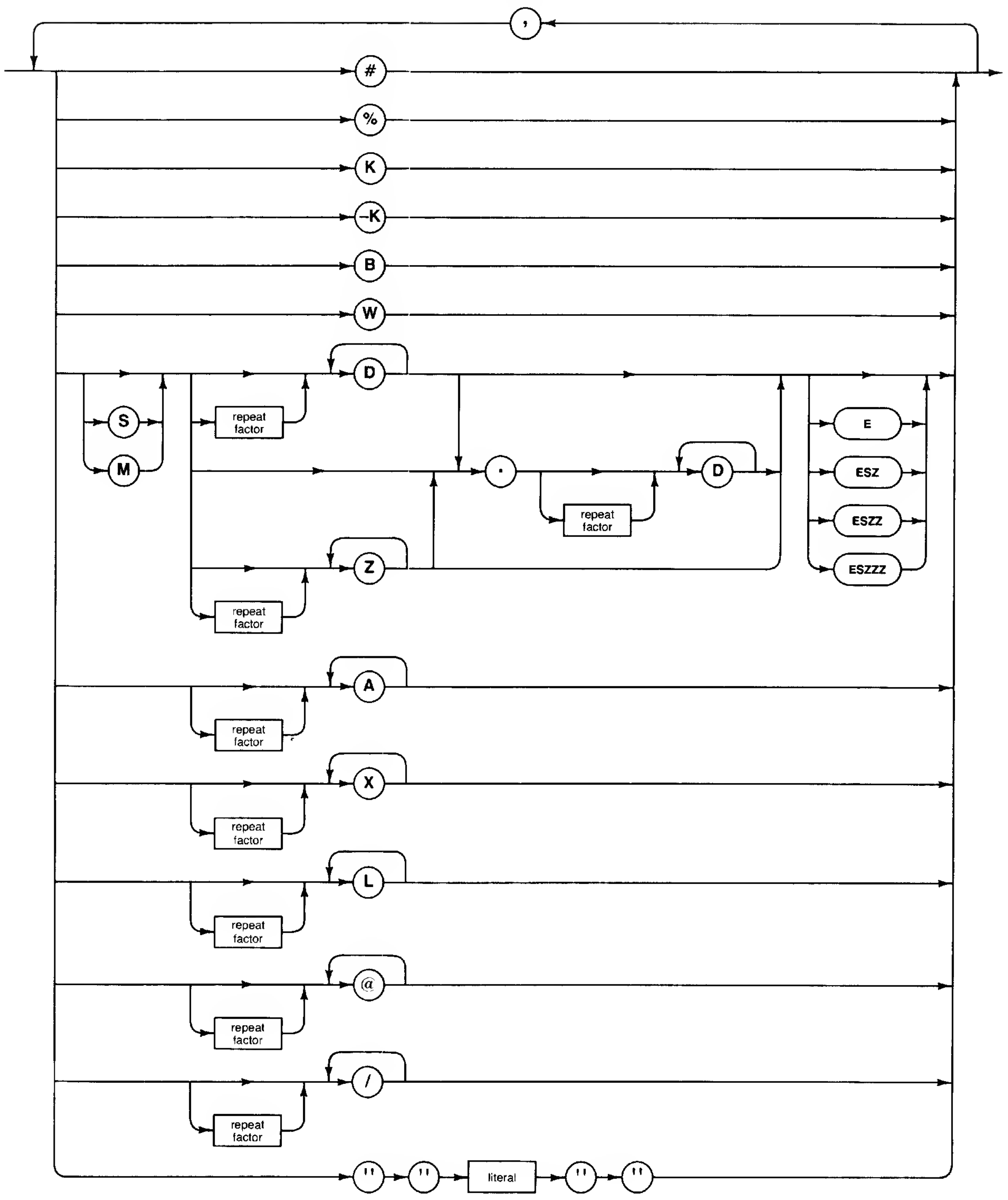


Item	Description/Default	Range Restrictions
image line number	integer constant identifying a program line	1 thru 32 766
image line label	name of a program line	any valid name
image specifier	string expression	(see drawing)
string array name	name of a string array	any valid name
numeric array name	name of a numeric array	any valid name
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

Example Statements

```
LABEL Number,String$
LABEL USING "5Z.DD";Money
```

image specifier list



Semantics

The label begins at the current logical pen position, with the current pen. Labels are clipped at the current clip boundary. Other statements which affect label generation are PEN, LINE TYPE, CSIZE, LORG, and LDIR. The current pen position is updated at the end of the label operation.

Standard Numeric Format

The standard numeric format depends on the value of the number being output. If the absolute value of the number is greater than or equal to 1E−4 and less than 1E+6, it is rounded to 12 digits and output in floating point notation. If it is not within these limits, it is output in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the label list is exhausted, an End-Of-Line (EOL) sequence is sent to the logical pen, unless it is suppressed by trailing punctuation or a pound-sign image specifier. The EOL sequence is also sent after every hundred characters. This "plotter buffer exceeded" EOL is not suppressed by trailing punctuation, but is suppressed by the pound-sign specifier.

Control Codes

Some ASCII control codes have a special effect in LABEL statements.

Character	Keystroke	Name	Action
CHR$(8)	CTRL-H	backspace	Back up the width of one character cell.
CHR$(10)	CTRL-J	linefeed	Move down the height of one character cell.
CHR$(13)	CTRL-M	carriage return	Move back the length of the label just completed.

Any control character that the LABEL statement does not recognize is treated as an ASCII blank [CHR$(32)].

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.

Note 2: The starting point for labels drawn after other labels is affected by LDIR.

Arrays

Arrays may be output as labels by using the asterisk specifier. They are output in row-major order (right-most subscript varies most rapidly) and their format depends on the label mode selected.

LABEL Without Using

If LABEL is used without USING, the punctuation following an item determines the width of the item's label field; a semicolon selects the compact field, and a comma selects the default label field. When the label item is an array with the asterisk array specifier, each array element is considered a separate label item. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the label field to be used for the label item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are output with one trailing blank. String items are output with no leading or trailing blanks.

The default label field labels items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is output with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

LABEL With Using

When the 9826 executes a LABEL USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the label items, the field specifier is acted upon without accessing the label list. When the field specifer requires characters, it accesses the next item in the label list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching label item. If the image specifiers are exhausted before the label items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than are specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

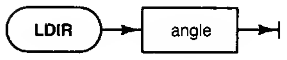
Effects of the image specifiers on a LABEL statement are shown in the following table.

Image Specifier	Meaning
K −K	Compact field. outputs a number or string as a label in standard form with no leading or trailing blanks.
S	Outputs the number's sign (+ or −) as a label.
M	Outputs the number's sign as a label if negative, a blank if positive.
D	Outputs one digit character as a label. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is labeled, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are output as a label.
B	Outputs as a label the character represented by one byte of data. This is similar to the CHR$ function. The least significant eight bits of the number are sent. The number is rounded to an integer. If the number is greater than 32 767, 255 is used; if the number is less than −32 768, 0 is used.
W	Outputs as a label the two characters represented by the two bytes in a 16-bit word. The number is rounded to an integer. If the number is larger than 32 767, 32 767 is used; if the number is less than −32 768, then −32 768 is used. The most-significant byte is output first, followed by the least-significant byte.
A	Outputs a string character as a label. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Outputs a blank as a label.
.	Outputs a decimal point radix indicator as a label.
E ESZZ	Outputs an E, a sign, and a two digit exponent as a label.
ESZ	Outputs an E, a sign, and a one digit exponent as a label.
ESZZZ	Outputs an E, a sign, and a three digit exponent as a label.
#	Suppresses all automatic output of the EOL (End-Of-Line) sequence.
%	Ignored in LABEL images.
L	Sends an EOL sequence.
@	Sends a form-feed; produces a blank.
/	Sends a carriage-return and a line-feed.
literal	Outputs the characters contained in the literal as a label.

LDIR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines the angle at which labels are drawn. The angle is interpreted as counter-clockwise, from horizontal. The current angle mode is used.

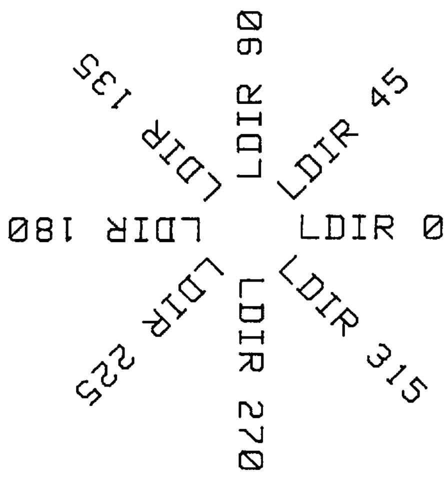


Item	Description/Default	Range Restrictions
angle	numeric expression in current units of angle; Default = 0	(same as COS)

Example Statements

```
LDIR 90
LDIR ACS(Side)
```

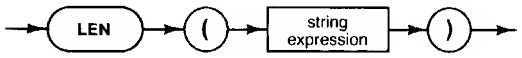
LDIR EXAMPLES (in Degrees)



LEN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the current number of characters in the argument. The length of the null string (" ") is 0.



Item	Description/Default	Range Restrictions
argument	string expression	—

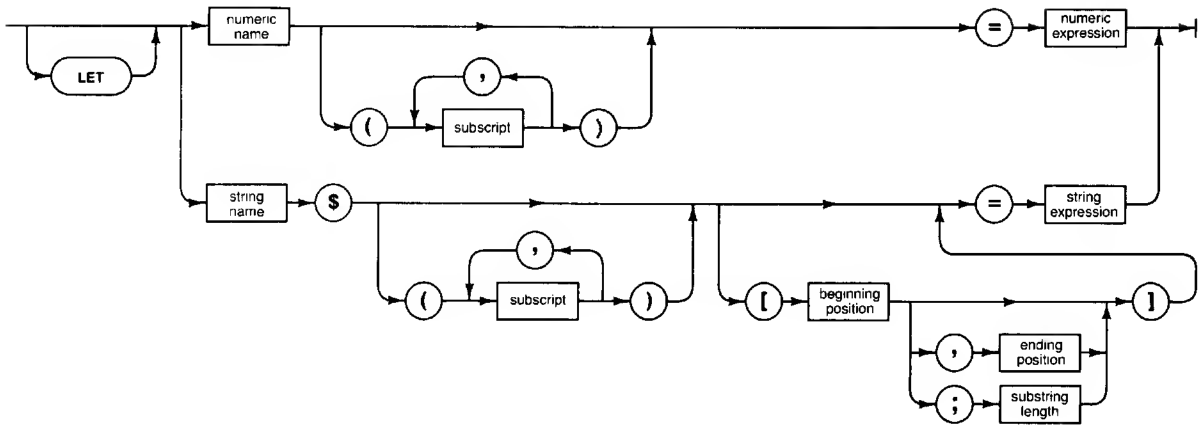
Example Statements

```
Last=LEN(String$)
IF NOT LEN(A$) THEN Empty
```

LET

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This is the assignment statement, which is used to assign values to variables.



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
LET Number=33
Array(I+1)=Array(I)/2
String$="Hello Scott"
A$(7)[1;2]=CHR$(27)&"Z"
```

Semantics

The assigment is done to the variable which is to the left of the equals sign. Only one assignment may be performed in a LET statement; any other equal signs are considered relational operators, and must be enclosed in a parenthetical expression (i.e. `A=A+(B=1)+5`). A variable can occur on both sides of the assignment operator (i.e. `I=I+1` or `Source$=Source$&Temp$`).

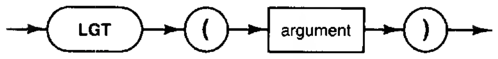
A real expression will be rounded when assigned to an INTEGER variable, if it is within the INTEGER range. Out-of-range assignments to an INTEGER give an error.

The length of the string expression must be less than or equal to the dimensioned length of the string it is being assigned to. Assignments may be made into substrings, using the normal rules for substring definition. The string expression will be truncated or blank-filled on the right (if necessary) to fit the destination substring when the substring has an explicitly stated length. If only the beginning position of the substring is specified, the expression must fit within the substring.

LGT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the logarithm (base 10) of its argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	greater than 0

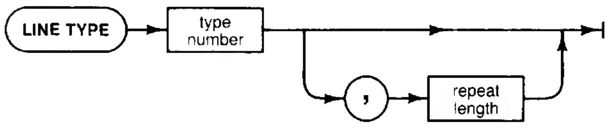
Example Statements

```
Decibel=20*LGT(Volts)
PRINT "Log of";X;"=";LGT(X)
```

LINE TYPE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a line type and repeat length for lines, labels, frames, axes and grids.



Item	Description/Default	Range Restrictions	Recommended Range
type number	numeric expression, rounded to an integer; Default = 1	1 thru 10	—
repeat length	numeric expression, rounded to an integer; Default = 5	−32 768 thru +32 767	greater than 0

Example Statements

```
LINE TYPE 1
LINE TYPE Select,20
```

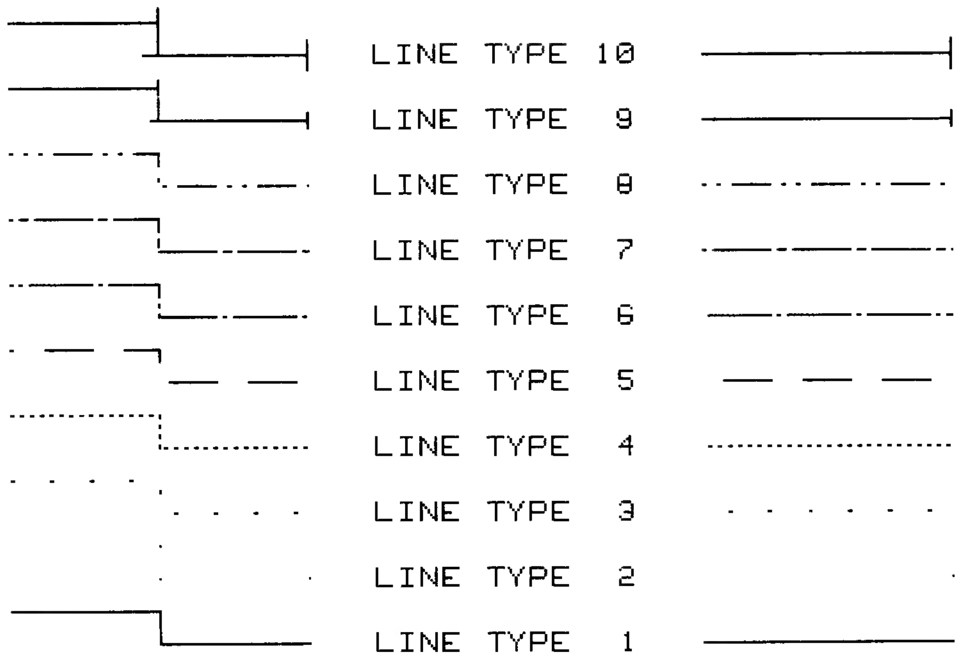
Semantics

At power-up the default line type is a solid line (type 1), and the default repeat length is 5 GDUs. While a negative pen number (erase) is selected, the line type used is not necessarily the most recently selected line type. If the most recent line type was 1 thru 8, erasures are done with line type 1. Line types 9 and 10 are erased with themselves. When a non-negative pen is selected, the line type is restored to the most recently selected line type.

The repeat length establishes the number of GDUs required to contain an arbitrary seqment of the line pattern. When the plotter is the internal CRT, the repeat length is evaluated and taken as the next lower multiple of 5, with a minimum value of 5.

When the plotter is an external plotter, the line produced by the line identifier is device dependent. Refer to your plotter's documentation for further information.

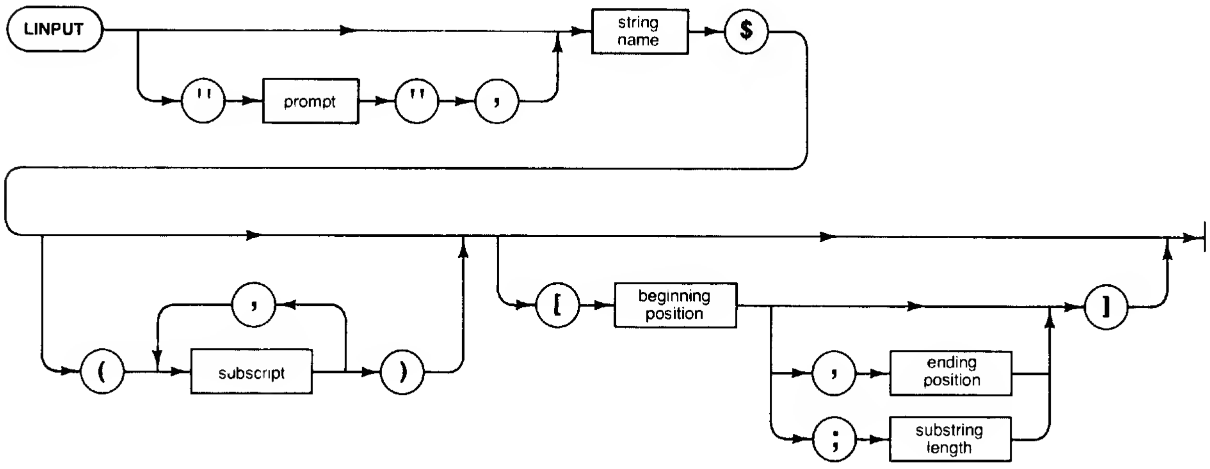
The available CRT line types are shown here.



LINPUT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement accepts alphanumeric input from the 9826 keyboard for assignment to a string variable. The LINPUT statement allows commas or quotation marks to be included in the value of the string, and leading or trailing blanks are not deleted.



Item	Description/Default	Range Restrictions
prompt	a literal composed of characters from the keyboard, including those generated using the ANY CHAR key; Default = question mark	—
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
LINPUT "Next Command?",Response$
LINPUT Array$(I)[3]
```

Semantics

A prompt, which remains until the LINPUT item is satisfied, appears on the CRT display line. If the last DISP statement suppressed its CR/LF, the prompt is appended onto the current display line contents. If the last DISP did not suppress the CR/LF, the prompt replaces the current display line contents. Not specifying a prompt results in the question mark being used as the prompt. Specifying the null string (" ") for the prompt suppresses the question mark.

(CONTINUE), (ENTER) or (STEP) must be pressed to indicate that the entry is complete. If no value is provided from the keyboard, the null string is used. If (CONTINUE) or (ENTER) is pressed to end the data input, program execution continues at the next program line. If (STEP) is pressed, the program execution continues at the next program line in single step mode. (If the LINPUT was stepped into, it is stepped out of, even if (CONTINUE) or (ENTER) is pressed.)

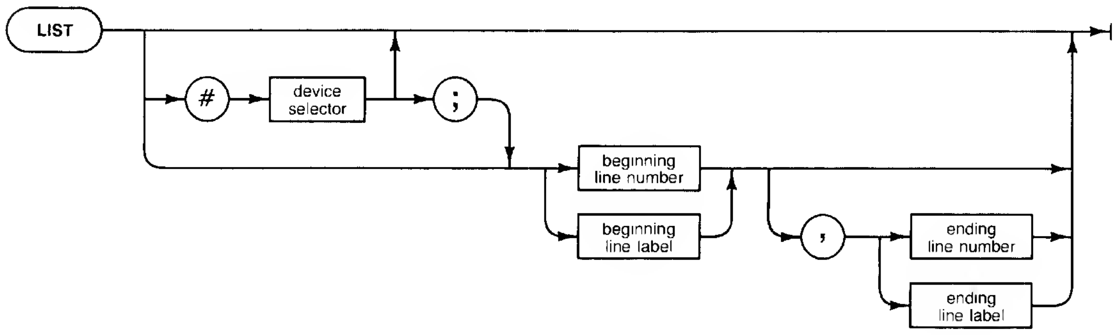
Live keyboard operations are not allowed while a LINPUT is waiting for data entry. (PAUSE) can be pressed so live keyboard operations can be performed. The LINPUT statement is re-executed from the beginning when (CONTINUE) or (STEP) is pressed.

ON KEY and ON KNOB events are deactivated during an LINPUT statement. Errors do not cause an ON ERROR branch. If an input response results in an error, the LINPUT statement is re-executed.

LIST

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lists the program currently in memory to the selected device. Beginning and ending line labels or numbers may be specified to list parts of the program.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer; Default = PRINTER IS device	(see Glossary)
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
LIST #701
LIST 110,250
```

Semantics

When a label is used as a line identifier, the lowest-numbered line in memory having the label is used. When a number is used as a line identifier, the lowest-numbered line in memory that has a number equal to or greater than the specified number is used.

Executing LIST from the keyboard while the program is running causes the program execution to pause at the end of the current program line. The listing is sent to the specified device, and program execution resumes.

The available memory in the 9826 is displayed after the listing is finished.

An error occurs if the ending line identifier occurs before the beginning line identifier or if a specified line label does not exist in the program.

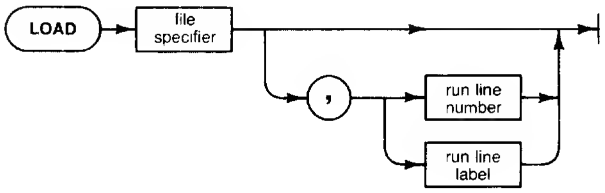
LISTEN

See the SEND statement.

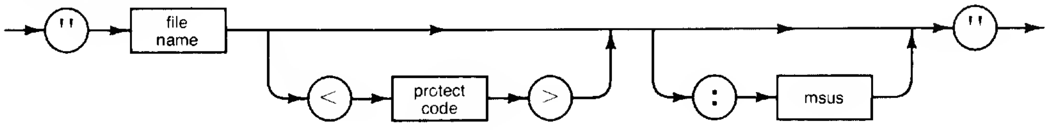
LOAD

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement loads PROG files into memory. PROG files are created by the STORE statement.



literal form of file specifier



protect code is ignored

Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	—
msus	literal; Default = MASS STORAGE IS device	INTERNAL
run line number	integer constant identifying a program line	1 thru 32 765
run line label	name of a program line	any valid name

Example Statements

```
LOAD "George"
LOAD Next_file$,500
```

Semantics

Any BASIC program, and all variables not in common are lost when LOAD is executed. If the COM area of the newly-loaded program does not match the existant COM area, the values in the old COM area are lost. Binary programs currently in the computer are preserved. If a PROG file contains a binary program with a name identical to the binary program in the computer, the new binary program is not loaded into memory.

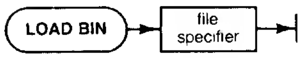
LOAD is allowed from the keyboard if a program is not running. If no run line is specified, RUN must be pressed to begin program execution. If a run line is specified, prerun initialization (see RUN) is performed and program execution begins at the specified line. The specified line must be in the main program context of the newly-loaded program.

Executing LOAD from a program causes a prerun, and program execution begins at either the specified run line or the lowest numbered program line in memory. If a run line is specified, it must be in the main program context of the newly-loaded program.

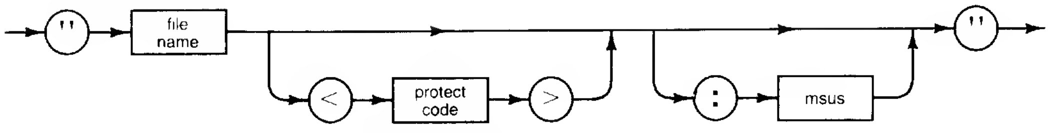
LOAD BIN

Keyboard Executable	Yes
Programmable	No

This command loads a BIN file into memory. BIN files are created with the STORE BIN statement.



literal form of file specifier:



protect code is ignored

Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL

Example Statements

```
LOAD BIN "BEB"
LOAD BIN Name$&Msus$
```

Semantics

Executing LOAD BIN does not affect either the currently loaded BASIC program or the values of any variables.

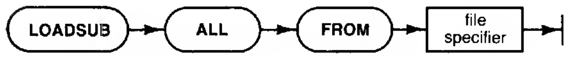
A BIN file may contain more than one binary program. Any binary program which is already in memory will not be loaded.

LOAD BIN may not be executed while a program is running. If LOAD BIN is executed while the computer is paused, the computer enters the stopped state.

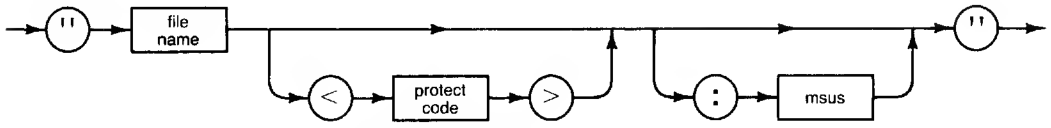
LOADSUB

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement loads BASIC subprograms in a file of type PROG into memory. Files of type PROG are created by the STORE statement.



literal form of file specifier:



protect code is ignored

Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

Example Statements

```
LOADSUB ALL FROM "George"
LOADSUB ALL FROM Name$&Msus$
```

Semantics

LOADSUB ALL FROM loads all the subprograms in a file into memory. If a subprogram in the file has the same name as a subprogram already in memory, it is loaded anyway. Both subprograms will be resident at the same time. The subprogram with the lowest beginning line number is used by CALL or FN, so the most recently loaded one is ignored.

LOADSUB does not:

- Affect the main program currently in the machine,
- Bring in any binary programs,
- Change the contents of any variables.

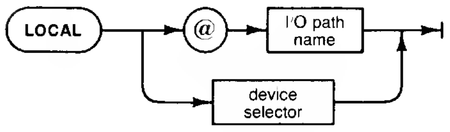
Subprograms brought into memory are renumbered as necessary.

If a LOADSUB is executed by a program, a prerun initialization (see RUN) of the **new** program segments is performed. Program execution resumes at the statement following the LOADSUB. Since this does not perform a **full** program prerun, any attempt to change the layout of the existing COM area generates a non-recoverable error. Executing a LOADSUB statement from the keyboard while a program is running pauses the program while the subprograms are loaded. The program resumes after the subprograms are loaded, as if the last statement executed had been a LOADSUB.

LOCAL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement returns all specified devices to their local state.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
LOCAL @Dvm
LOCAL 7
```

Semantics

If only an interface select code is specified by the I/O path name or device selector, all devices on the bus are returned to their local state by setting REN false. Any existing LOCAL LOCKOUT is cancelled.

If a primary address is included, the GTL message (Go To Local) is sent to all listeners. LOCAL LOCKOUT is not cancelled.

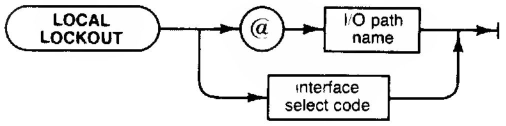
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	$\overline{\text{REN}}$ ATN	ATN MTA UNL LAG GTL	ATN GTL	ATN MTA UNL LAG GTL
Not Active Controller	$\overline{\text{REN}}$ ATN	Error	Error	

LOCAL LOCKOUT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This HP-IB statement sends the LLO (local lockout) message, preventing an operator from returning the specified device to local (front panel) control.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
LOCAL LOCKOUT 7
LOCAL LOCKOUT @Hpib
```

Semantics

The computer must be the active controller to execute LOCAL LOCKOUT.

If a device is in the LOCAL state when this message is sent, it does not take effect on that device until the device receives a REMOTE message and becomes addressed to listen.

LOCAL LOCKOUT does not cause bus reconfiguration, but issues a universal bus command received by all devices on the interface whether addressed or not. The command sequence is ATN and LLO.

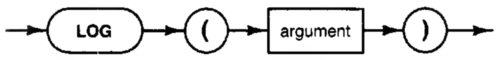
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN LLO	Error	ATN LLO	Error
Not Active Controller	Error			

LOG

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the natural logarithm (base e) of the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	greater than 0

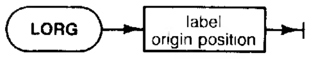
Example Statements

```
Time=-1*Rc*LOG(Volts/Emf)
PRINT "Natural log of";Y;"=";LOG(Y)
```

LORG

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the relative origin of labels with respect to the current pen position.



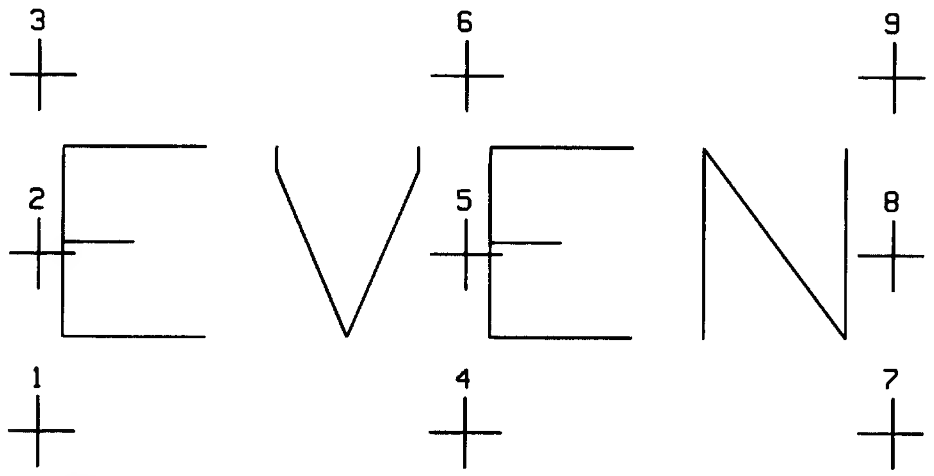
Item	Description/Default	Range Restrictions
label origin position	numeric expression, rounded to an integer; Default = 1	1 thru 9

Example Statements

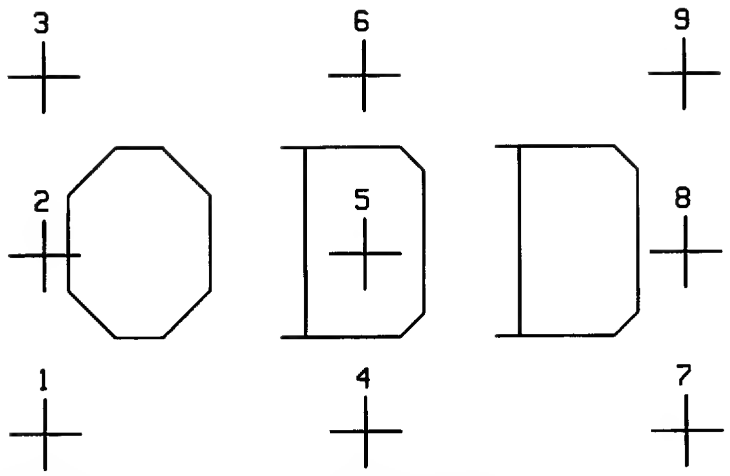
```
LORG 4
IF Y>Limit THEN LORG 3
```

Semantics

The following drawings show the relationship between a label and the logical pen position. The pen position before the label is drawn is represented by a cross marked with the appropriate LORG number.



Label Origins for Labels with an Even Number of Characters

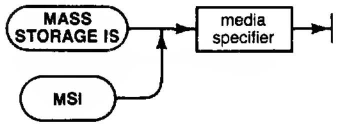


Label Origins for Labels with an Odd Number of Characters

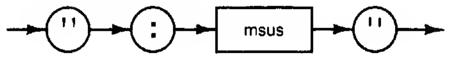
MASS STORAGE IS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the system mass storage device.



literal form of media specifier:



Item	Description/Default	Range Restrictions
media specifier	string expression	(see drawing)
msus	literal	INTERNAL

Example Statements

```
MASS STORAGE IS ":INTERNAL"
MASS STORAGE IS Msus$
```

Semantics

All mass storage operations which do not specify a source or destination by either an I/O path name or msus in the file specifier use the current system mass storage device.

MASS STORAGE IS can be abbreviated as MSI when entering a program line, but MSI is always listed in a program as MASS STORAGE IS.

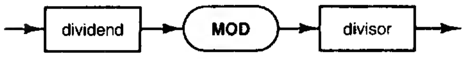
MLA

See the SEND statement.

MOD

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns the remainder of an integer division.



Item	Description/Default	Range Restrictions
dividend	numeric expression	—
divisor	numeric expression	not equal to 0

Example Statements

```
Remainder=Dividend MOD Divisor
PRINT "Seconds =";Time MOD 60
```

Semantics

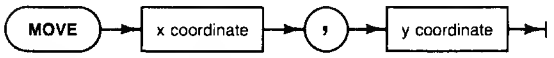
MOD returns an INTEGER value if both arguments are INTEGER. Otherwise the returned value is REAL.

MOD is equivalent to X − Y × (X DIV Y). This may return a different result from the modulus function on other computers when negative numbers are involved.

MOVE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement updates the logical pen position.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression in current units	—
y coordinate	numeric expression in current units	—

Example Statements

```
MOVE 10,75
MOVE Next_x,Next_y
```

Semantics

The actual pen is not moved until an operation is performed which requires the pen to draw something. The logical pen may bear no obvious relation to the actual pen.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

MTA

See the SEND statement.

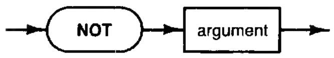
NEXT

See the FOR...NEXT statement.

NOT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns 1 if its argument equals 0. Otherwise, 0 is returned.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

Example Statements

```
Invert_flag=NOT Std_device
IF NOT Pointer THEN Next_op
```

Semantics

When evaluating the argument, a non-zero value (positive or negative) is treated as a logical 1; only zero is treated as a logical 0.

The logical complement is shown below:

A	NOT A
0	1
1	0

NPAR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the number of parameters passed to the current subprogram. If execution is currently in the main program, NPAR returns 0.

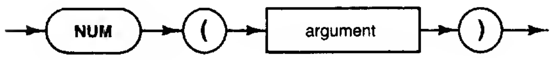
Example Statements

```
IF NPAR>3 THEN Extra
Factors=NPAR-2
```

NUM

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the decimal value of the ASCII code of the first character in the argument. The range of returned values is 0 thru 255.



Item	Description/Default	Range Restrictions
argument	string expression	not a null string

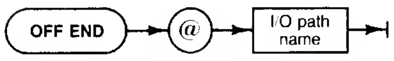
Example Statements

```
Letter=NUM(String$)
A$[I;1]=CHR$(NUM(A$[I])+32)
```

OFF END

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously enabled and defined by an ON END statement.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)

Example Statements

```
OFF END @File
IF Special THEN OFF END @Source
```

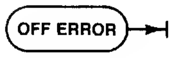
Semantics

If OFF END is executed in a subprogram and cancels an ON END in the context which called the subprogram, the ON END definitions are restored when the calling context is restored.

OFF ERROR

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

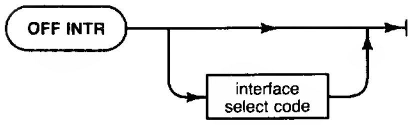
This statement cancels event-initiated branches previously defined and enabled by an ON ERROR statement. Further errors are reported to the user in the usual fashion.



OFF INTR

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined by an ON INTR statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer; Default = all interfaces	7 thru 31

Example Statements

```
OFF INTR
OFF INTR Hpib
```

Semantics

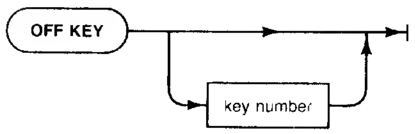
Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the OFF INTR to apply to the event-initiated log entry for the specified interface only.

Any pending ON INTR branches for the effected interfaces are lost and further interrupts are ignored.

OFF KEY

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON KEY statement.



Item	Description/Default	Range Restrictions
key number	numeric expression, rounded to an integer; Default = all keys	0 thru 19

Example Statements

```
OFF KEY
OFF KEY 4
```

Semantics

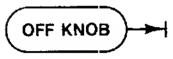
Not specifying a softkey number disables the event-initiated branches for all softkeys. Specifying a softkey number causes the OFF KEY to apply to the specified softkey only. If OFF KEY is executed in a subprogram and cancels an ON KEY in the context which called the subprogram, the ON KEY definitions are restored when the calling context is restored.

Any pending ON KEY branches for the effected softkeys are lost. Pressing an undefined softkey generates a beep.

OFF KNOB

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

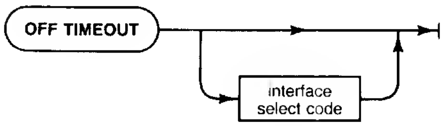
This statement cancels event-initiated branches previously defined and enabled by the ON KNOB statement. Any pending ON KNOB branches are lost. Further use of the knob will result in normal scrolling or cursor movement.



OFF TIMEOUT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels event-initiated branches previously defined and enabled by an ON TIMEOUT statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer; Default = all interfaces	7 thru 31

Example Statements

```
OFF TIMEOUT
OFF TIMEOUT Isc
```

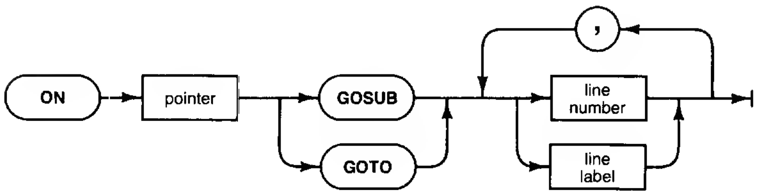
Semantics

Not specifying an interface select code disables the event-initiated branches for all interfaces. Specifying an interface select code causes the ON TIMEOUT to apply to the event-initiated branches for the specified interface only. When OFF TIMEOUT is executed, no more timeouts can occur on the effected interfaces.

ON

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement transfers program execution to one of several destinations selected by the value of the pointer.



Item	Description/Default	Range Restrictions
pointer	numeric expression, rounded to an integer	1 thru 45
line number	integer constant identifying a program line	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
ON X1 GOTO 100,150,170
IF Point THEN ON Point GOSUB First,Second,Third,Last
```

Semantics

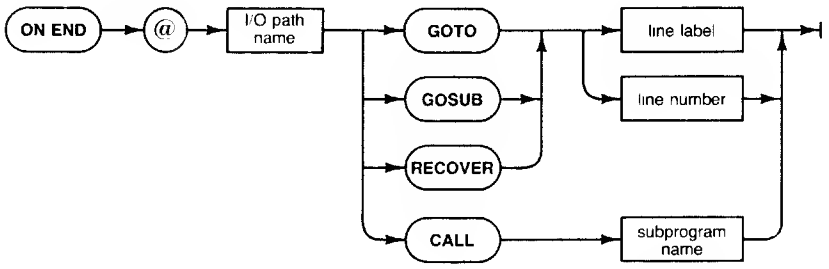
If the pointer is 1, the first line number or label is used. If the pointer is 2, the second line identifier is used, and so on. If GOSUB is used, the RETURN is to the line following the ON...GOSUB statement.

If the pointer is less than 1 or greater than the number of line labels or numbers, error 19 is generated. The specified line numbers or line labels must be in the same context as the ON statement.

ON END

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch to be taken when end-of-file is reached on the mass storage file associated with the specified I/O path.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a mass storage file	any valid name (see ASSIGN)
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON END @Source GOTO Next_file
ON END @Dest CALL Expand
```

Semantics

The ON END branch is triggered by any of the following events:

- When the physical end-of-file is encountered.
- When an ENTER statement reads the byte at EOF or beyond.
- When an invalid record number is specified by a random access ENTER or OUTPUT.
- When a random access OUTPUT requires more than one defined record.
- When a random access OUTPUT is attempted beyond the next available record. (If EOF is the first byte of a record, then that record is the next available record. If EOF is not at the first byte of a record, the following record is the next available record.)

The priority associated with ON END is higher than priority 15. ON TIMEOUT and ON ERROR have the same priority as ON END, and can interrupt an ON END service routine.

Any specified line label or line number must be in the same context as the ON END statement. CALL and GOSUB will return to the line immediately following the one during which the end-of-file occurred. RECOVER forces the program to go directly to the specified line in the context containing the ON END statement.

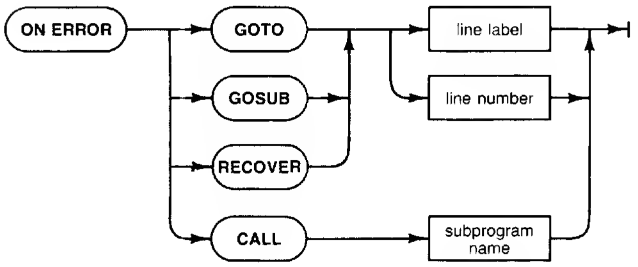
CALL and RECOVER remain active when the context changes to a subprogram, if the I/O path name is known in the new context. CALL and RECOVER do not remain active if the context changes as a result of a keyboard-originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram.

The end-of-record error (error 60) or the end-of-file error (error 59) can be trapped by ON ERROR if ON END is not active. ON END is deactivated by OFF END. DISABLE does not affect ON END.

ON ERROR

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch which results from a trappable error. This allows you to write your own error handling routines.



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON ERROR GOTO 1200
ON ERROR CALL Report
```

Semantics

The ON ERROR statement has the highest priority of any event-initiated branch. ON ERROR can interrupt any event-initiated service routine.

Any specified line label or line number must be in the same context as the ON END statement. RECOVER forces the program to go directly to the specified line in the context containing the ON END statement.

Returns from ON ERROR GOSUB or ON ERROR CALL routines are different from regular GOSUB or CALL returns. When ON ERROR is in effect, the program resumes at the beginning of the line where the error occurred. If the ON ERROR routine did not correct the cause of the error, the error is repeated. This causes an infinite loop between the line in error and the error handling routine.

CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. In this case, the error is reported to the user, as if ON ERROR had not been executed.

GOSUB and GOTO do not remain active when the context changes to a subprogram. If an error occurs, the error is reported to the user, as if ON ERROR had not been executed.

If an execution error occurs while servicing an ON ERROR CALL or ON ERROR GOSUB, program execution stops. If an execution error occurs while servicing an ON ERROR GOTO or ON ERROR RECOVER routine, an infinite loop can occur between the line in error and the GOTO or RECOVER routine.

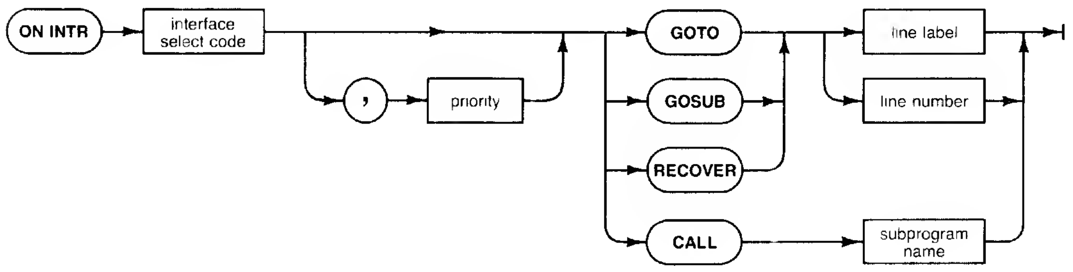
If an ON ERROR routine cannot be serviced because inadequate memory is available for the computer, the original error is reported and program execution pauses at that point.

ON ERROR is deactivated by OFF ERROR. DISABLE does not affect ON ERROR.

ON INTR

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines an event-initiated branch to be taken when an interface card generates an interrupt. The interrupts must be explicitly enabled with an ENABLE INTR statement.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON INTR 7 GOSUB 500
ON INTR Isc,4 CALL Service
```

Semantics

The occurrence of an interrupt performs an implicit DISABLE INTR for the interface. An ENABLE INTR must be performed to re-enable the interface for subsequent event-initiated branches. Another ON INTR is not required, nor must the mask for ENABLE INTR be redefined.

The priority can be specified, with highest priority represented by 15. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON INTR can interrupt other ON INTR, ON KNOB, or ON KEY service routines if the ON INTR priority is higher than the priority of the service routine. CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

CALL and GOSUB will return to the next line that would have been executed if the interrupt had not been serived. RECOVER forces the program to go directly to the specified line in the context containing the ON INTR statement.

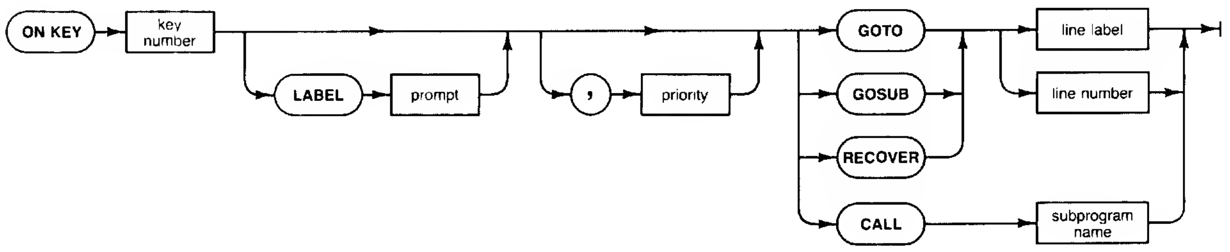
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON INTR is disabled by DISABLE INTR or DISABLE and deactivated by OFF INTR.

ON KEY

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch which occurs when a softkey is pressed.



Item	Description/Default	Range Restrictions
key number	numeric expression, rounded to an integer	0 thru 19
prompt	string expression	—
priority	numeric expression, rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON KEY 0 GOTO 150
ON KEY 5 LABEL "Print",3 GOSUB Report
```

Semantics

The most recently executed ON KEY definition (and label) for a particular softkey overrides any previous key definition, except when changing contexts.

Labels appear at the two bottom lines of the CRT. When a subprogram is invoked, the labels are transferred into the new context. If a subprogram contains ON KEY definitions of its own, the old labels and branch definitions are saved, and restored when the calling context is restored.

The priority can be specified, with highest priority represented by 15. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON KEY can interrupt other ON INTR, ON KNOB, or ON KEY service routines if the ON KEY priority is higher than the priority of the service routine. CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

CALL and GOSUB will return to the next line that would have been executed if the interrupt had not been serviced. RECOVER forces the program to go directly to the specified line in the context containing the ON KEY statement.

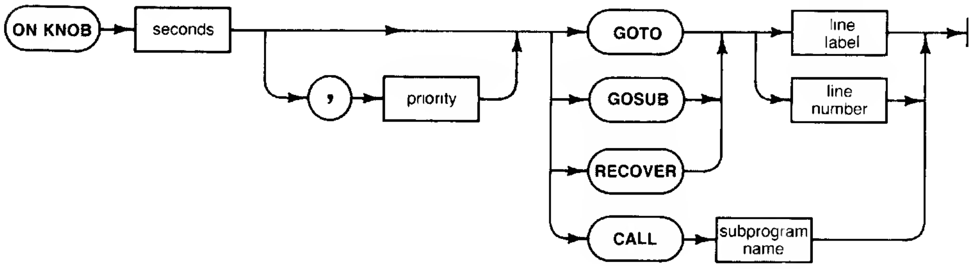
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON KEY is disabled by DISABLE, deactivated by OFF KEY, and temporarily deactivated when the program is paused or executing LINPUT, INPUT, or ENTER 2.

ON KNOB

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables event-initiated branches which result from turning the knob.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest hundredth	0.01 thru 2.55
priority	numeric expression rounded to an integer; Default = 1	1 thru 15
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON KNOB .1 GOSUB 250
ON KNOB .333,Priority CALL Pulses
```

Semantics

Turning the knob (cursor wheel) generates pulses. After ON KNOB is activated (or re-activated), the first pulse received starts a sampling interval. The "seconds" parameter establishes the length of that sampling interval. At the end of the sampling interval, the ON KNOB branch is taken if the net number of pulses received during the interval is not zero. The KNOBX function can be used to determine the number of pulses received during the interval. If the ON KNOB branch is held off for any reason, the KNOBX function accumulates the pulses (see KNOBX).

The priority can be specified, with highest priority represented by 15. The highest priority is less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON KNOB can interrupt other ON INTR, ON KNOB, or ON KEY service routines if the ON KNOB priority is higher than the priority of the service routine. CALL and GOSUB service routines get the priority specified in the ON... statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

Any specified line label or line number must be in the same context as the ON KNOB statement. CALL and GOSUB will return to the next line that would have been executed if the interrupt had not been serviced. RECOVER forces the program to go directly to the specified line in the context containing the ON KNOB statement.

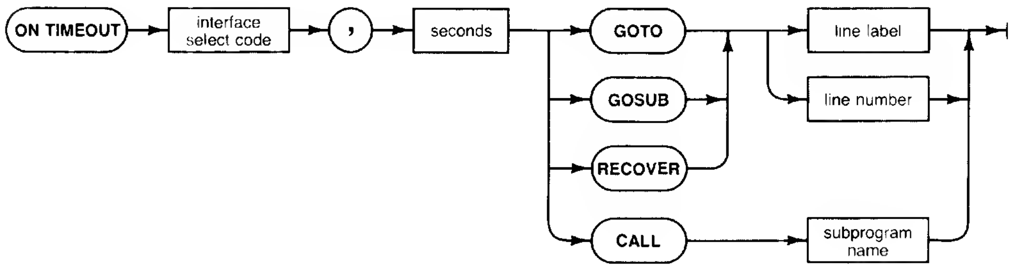
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard originated call. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch cannot be taken until the calling context is restored.

ON KNOB is disabled by DISABLE and canceled by OFF KNOB.

ON TIMEOUT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines and enables an event-initiated branch which results from an I/O timeout on the specified interface.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
seconds	numeric expression, rounded to the nearest thousandth	0.001 thru 32.767
line label	name of a program line	any valid name
line number	integer constant identifying a program line	1 thru 32 766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON TIMEOUT 7,4 GOTO 770
ON TIMEOUT Printer,Time GOSUB Message
```

Semantics

There is no default system timeout. If ON TIMEOUT is not in effect for an interface, a device can cause the program to wait forever.

The specified branch occurs if an input or output is active on the interface and the interface has not responded within the number of seconds specified. This time limit is approximate within ±25%.

Timeouts apply to ENTER and OUTPUT statements, and operations involving the PRINTER IS, PRINTALL IS, and PLOTTER IS devices when they are external. Timeouts do not apply to CONTROL, STATUS, READIO, WRITEIO, CRT alpha or graphics I/O, real time clock I/O, keyboard I/O, or mass storage operations.

The priority associated with ON TIMEOUT is higher than priority 15. ON END and ON ERROR have the same priority as ON TIMEOUT, and can interrupt an ON TIMEOUT service routine.

Any specified line label or line number must be in the same context as the ON TIMEOUT statement. CALL and GOSUB will return to the line immediately following the one during which the timeout occurred. RECOVER forces the program to go directly to the specified line in the context containing the ON TIMEOUT statement.

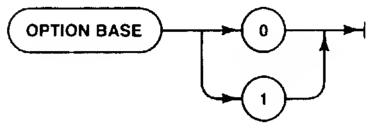
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard originated call. GOSUB and GOTO do not remain active when the context changes to a subprogram.

ON TIMEOUT is canceled by OFF TIMEOUT. DISABLE does not affect ON TIMEOUT.

OPTION BASE

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the default lower bound of arrays.



Example Statements

```
OPTION BASE 0
OPTION BASE 1
```

Semantics

This statement can occur only once in each context. If used, OPTION BASE must precede any explicit variable declarations in a context. Since arrays are passed to subprograms by reference, they maintain their orginal lower bound, even if the new context has a different OPTION BASE. Any context that does not contain an OPTION BASE statement assumes default lower bounds of zero.

The OPTION BASE value is determined at prerun, and is used with all arrays declared without explicit lower bounds in COM, DIM, INTEGER, and REAL statements as well as with all implicitly dimensioned arrays. OPTION BASE is also used at runtime for any arrays declared without lower bounds in ALLOCATE.

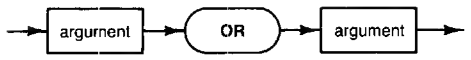
OPTIONAL

See DEF FN and SUB statements.

OR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This operator returns a 1 or a 0 based on the logical inclusive-or of the arguments.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

Example Statements

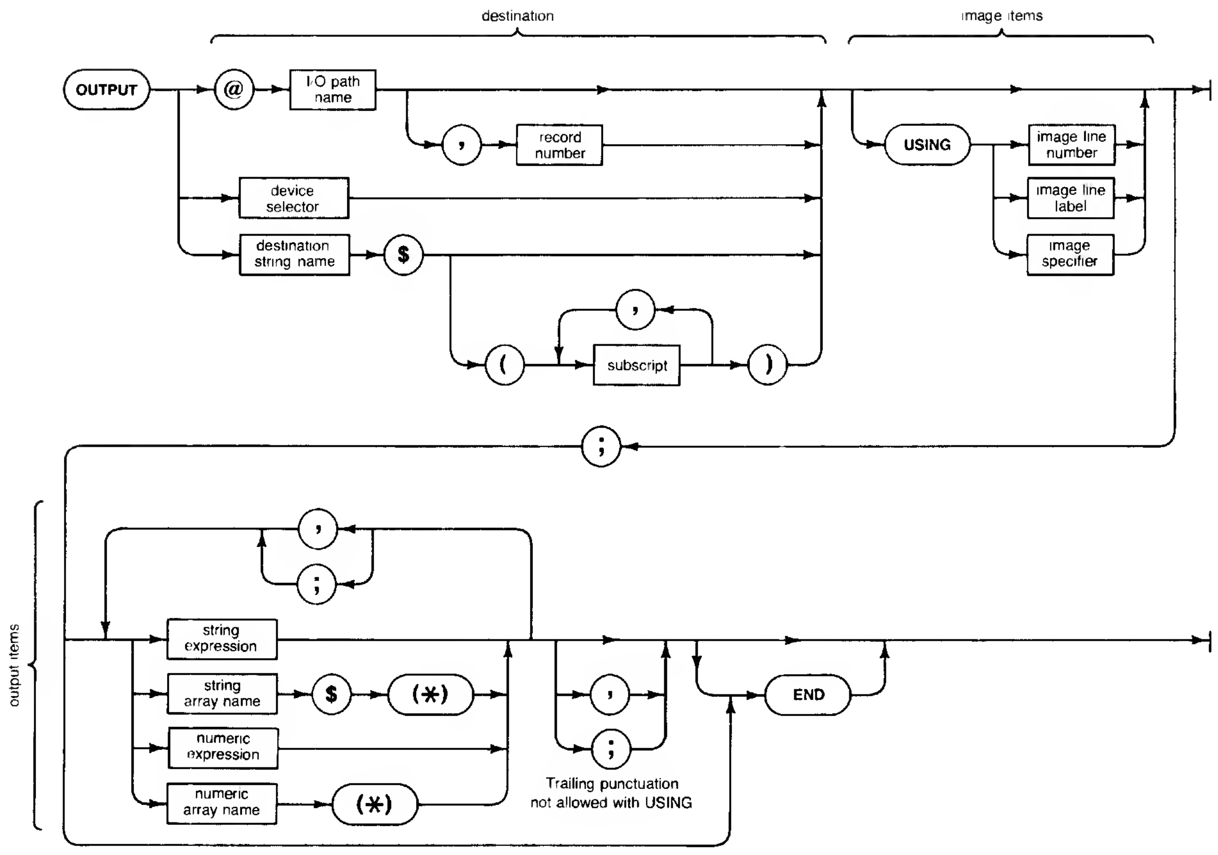
```
X=Y OR Z
IF File_type OR Device THEN Process
```

Semantics

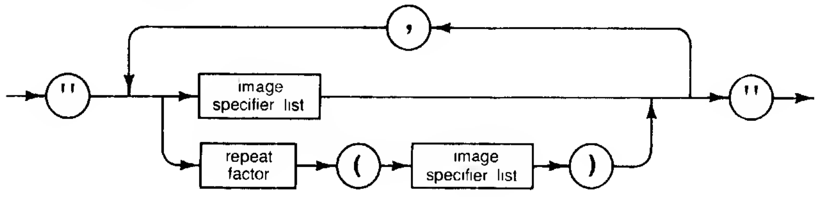
An expression which evaluates to a non-zero value is treated as a logical 1. An expression must evaluate to zero to be treated as a logical 0.

The truth table is:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



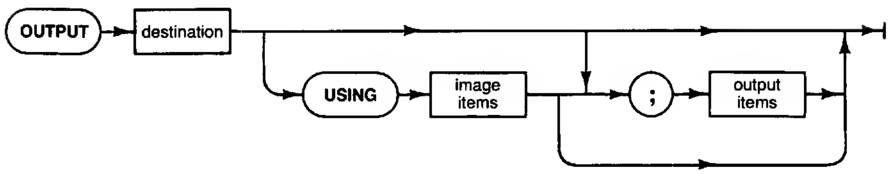
literal form of image specifier



OUTPUT

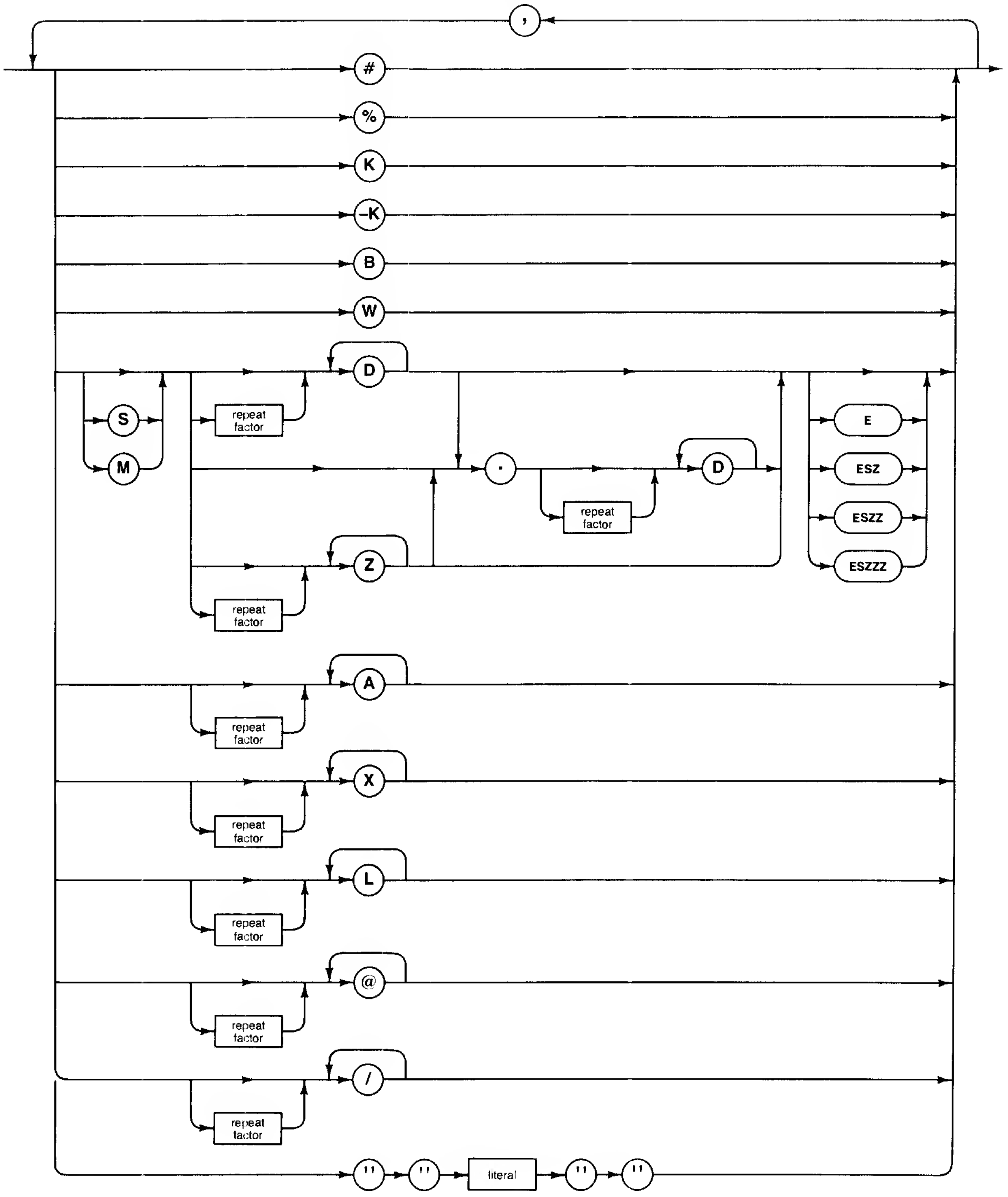
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement outputs items to a specified destination.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, or mass storage file	any valid name
record number	numeric expression, rounded to an integer	1 thru $2^{31}-1$
device selector	numeric expression, rounded to an integer	(see Glossary)
destination string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
image line number	integer constant identifying a program line	1 thru 32 766
image line label	name of a program line	any valid name
image specifier	string expression	(see drawing)
string array name	name of a string array	any valid name
numeric array name	name of a numeric array	any valid name
image specifier list	literal	(see next drawing)
repeat factor	integer constant	1 thru 32 767
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed

image specifier list



Example Statements

```
OUTPUT 701;Number,String$;
OUTPUT @File;Array(*),END
OUTPUT @Rand,5 USING Fmt1;Item(5)
OUTPUT 12 USING "#,6A";B$[2;6]
OUTPUT @Printer;Rank;Id;Name$
```

Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E−4 and less than 1E+6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Arrays

Entire arrays may be output by using the asterisk specifier. Each element in an array is treated as an item by the OUTPUT statement, as if the items were listed separately, separated by the punctuation following the array specifier. If no punctuation follows the array specifier, a comma is assumed. The array is output in row major order (rightmost subscript varies fastest.)

Files as Destination

If an I/O path has been assigned to a file, the file may be written to with OUTPUT statements. The file must be an ASCII or BDAT file. The attributes specified in the ASSIGN statement are used if the file is a BDAT file.

Serial access is available for both ASCII and BDAT files. Random access is available for BDAT files. The end-of-file marker (EOF) and the file pointer are important to both serial and random access. The file pointer is set to the beginning of the file when the file is opened by an ASSIGN. The file pointer always points to the next byte to be written by OUTPUT operations. The EOF pointer is read from the media when the file is opened by an ASSIGN. On a newly-created file, EOF is set to the beginning of the file. After each OUTPUT operation, the EOF is updated **internally** to the maximum of the file pointer or the previous EOF value. The EOF pointer is updated on the **media** at the following times:

- When the current end-of-file changes.
- When END is specified in an OUTPUT statement directed to the file.
- When a CONTROL statement directed to the I/O path name changes the position of the EOF.

Random access uses the record number parameter to write items to a specific location in a file. The OUTPUT begins at the start of the specified record and must fit into one record. The record specified cannot be beyond the record containing the EOF, if EOF is at the first byte of a record. The record specified can be one record beyond the record containing the EOF, if EOF is not at the first byte of a record. Random access is always allowed to records preceding the EOF record. If you wish to write randomly to a newly created file, either use a CONTROL statement to position the EOF in the last record, or write some "dummy" data into every record.

When data is written to an ASCII file, each item is sent as an ASCII representation with a 2-byte length header. Data sent to a BDAT file is sent in internal format if FORMAT is OFF, and is sent as ASCII characters if FORMAT is ON. (See "Devices as Destination" for a description of these formats.)

Devices as Destination

An I/O path or a device selector may be used to direct OUTPUT to a device. If a device selector is used, the default system attributes are used (see ASSIGN). If an I/O path is used, the ASSIGN statement used to associate the I/O path with the device also determines the attributes used. If multiple listeners were specified in the ASSIGN, the OUTPUT is directed to all of them. If FORMAT ON is the current attribute, the items are sent in ASCII. Items followed by a semicolon are sent with nothing following them. Numeric items followed by a comma are sent with a comma following them. String items followed by a comma are sent with a CR-LF following them. If the last item in the OUTPUT statement has no punctuation following it, the current end-of-line sequence (EOL) is sent after it. Trailing punctuation eliminates the automatic EOL.

If FORMAT OFF is the current attribute, items are sent to the device in the 9826's internal format. Punctuation following items has no effect on the OUTPUT. Two bytes are sent for each INTEGER, eight bytes for each REAL. Each string output consists of a four byte header containing the length of the string, followed by the actual string characters. If the number of characters is odd, an additional byte containing a blank is sent after the last character.

CRT as Destination

If the device selector is 1, the OUTPUT is directed to the CRT. OUTPUT 1 and PRINT differ in their treatment of separators and print fields. OUTPUT 1 USING and PRINT USING to the CRT produce similar actions.

Keyboard as Destination

Outputs to device selector 2 may be used to simulate keystrokes. ASCII characters can be sent directly (i.e. "hello"). Non-ASCII keys (such as EXECUTE) are simulated by a two byte sequence. The first byte is CHR$(255) and the second byte can be found in the Keycode Diagram in the back of this book.

When simulating keystrokes, unwanted characters (such as the EOL sequence) can be avoided with an image specifier (such as "#,B" or "#,K"). See "OUTPUT with USING".

Strings as Destination

If a string name is used for the destination, the string is treated similarly to a file. However, there is no file pointer; each OUTPUT begins at the beginning of the string, and writes serially within the string.

OUTPUT With USING

When the 9826 executes an OUTPUT USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the output items, the field specifier is acted upon without accessing the output list. When the field specifier requires characters, it accesses the next item in the output list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching output item. If the image specifiers are exhausted before the output items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifier, trailing blanks are used to fill out the field.

Effects of the image specifiers on an OUTPUT statement are shown below:

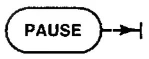
Image Specifier	Meaning
K −K	Compact field. Outputs a number or string in standard form with no leading or trailing blanks.
S	Outputs the number's sign (+ or −).
M	Outputs the number's sign if negative, a blank if positive.
D	Outputs one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is output, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are output.
B	Outputs the character represented by one byte of data. This is similar to the CHR$ function. The least significant eight bits of the number are sent. The number is rounded to an integer. If the number is greater than 32 767, 255 is used; if the number is less than −32 768, 0 is used.
W	Outputs two characters represented by the two bytes in a 16 bit word. The number is rounded to an integer. If the number is larger than 32 767, 32 767 is used; if the number is less than −32 768, then −32 768 is used. On an 8-bit interface, the most significant byte is sent first, followed by the least significant byte.
A	Outputs a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Outputs a blank.
.	Outputs a decimal point radix indicator.

Image Specifier	Meaning
E ESZZ	Outputs an E, a sign, and a two digit exponent.
ESZ	Outputs an E, a sign, and a one digit exponent.
ESZZZ	Outputs an E, a sign, and a three digit exponent.
#	Suppresses automatic output of the EOL (End-Of-Line) sequence at the end of the output list.
%	Ignored in PRINT images.
L	Outputs an EOL sequence.
@	Outputs a form-feed.
/	Outputs a carriage-return and a line-feed.
literal	Outputs the characters contained in the literal.

PAUSE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement suspends program execution.



Semantics

PAUSE suspends program execution before the next line is executed, until the CONTINUE key is pressed or CONT is executed. If the program is modified while paused, RUN must be used to restart program execution.

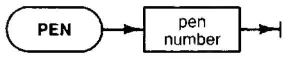
When program execution resumes, the computer attempts to service any ON INTR events that occurred while the program was paused. ON END, ON ERROR, or ON TIMEOUT events generate errors if they occur while the program is paused. ON KEY and ON KNOB events are ignored while the program is paused.

Pressing the PAUSE key, or typing `PAUSE` and pressing EXECUTE will suspend program execution at the end of the line currently being executed.

PEN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a pen on the current plotting device.



Item	Description/Default	Range Restrictions	Recommended Range
pen number	numeric expression, rounded to an integer; Default = 1	−32 768 thru +32 767	−1 thru 8 (device dependent)

Example Statements

```
PEN -1
PEN Select
```

Semantics

On an external plotter, no checking is done to verify if the pen actually exists on the plotter.

When the CRT is the plotter, any positive value is treated as PEN 1 and draws lines; any negative value is treated as PEN −1 and erases lines. PEN 0 complements whatever it passes over (draws lines where there are none, and erases lines where they exist).

PENUP

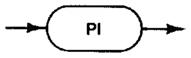
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement lifts the pen on an external plotter.

PI

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns 3.141 592 653 589 79, which is an approximate value for π.



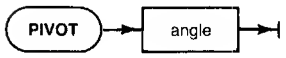
Example Statements

```
Area=PI*Radius^2
PRINT X,X*2*PI
```

PIVOT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies a rotation of coordinates which is applied to all lines drawn with DRAW and IDRAW statements.



Item	Description/Default	Range Restrictions
angle	numeric expression in current units of angle	(same as COS)

Example Statements

```
PIVOT 30
IF Special THEN PIVOT Radians
```

Semantics

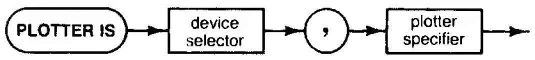
The angle is interpreted according to the current angle mode (DEG or RAD). The rotation is performed about the logical pen position at the time PIVOT is executed. The logical pen position may bear no obvious relationship to the physical pen position.

The PIVOT operation is applied to **drawn lines**, not to graphics statements. Therefore, it cannot be said that PIVOT does or does not affect MOVE. Lines that are drawn with a combination of MOVE and DRAW are effected by PIVOT. Labels that are created with a combination of MOVE and LABEL are **not** effected by PIVOT.

PLOTTER IS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a plotting device.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)
plotter specifier	string expression	INTERNAL HPGL

Example Statements

```
PLOTTER IS 3,I$
PLOTTER IS 705,"HPGL"
```

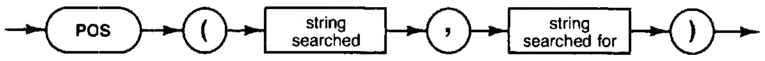
Semantics

The hard clip limits of the plotter are read in when this statement is executed. The PLOTTER IS device is 3, "INTERNAL" at power-on, after GINIT, after reset, and after SCRATCH A.

POS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the first position of a substring within a string.



Item	Description/Default	Range Restrictions
string searched	string expression	—
string searched for	string expression	—

Example Statements

```
Point=POS(Big$,Little$)
IF POS(A$,CHR$(10)) THEN Line_end
```

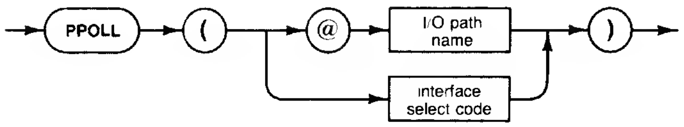
Semantics

If the value returned is greater than 0, it represents the position of the first character of the string being searched for in the string being searched. If the value returned is 0, the string being searched for does not exist in the string being searched (or the string searched for is the null string).

PPOLL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a value representing eight status-bit messages of devices on the HP-IB.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	7 thru 31

Example Statements

```
Stat=PPOLL(7)
IF BIT(PPOLL(@Hpib),3) THEN Respond
```

Semantics

The computer must be the active controller to execute this statement.

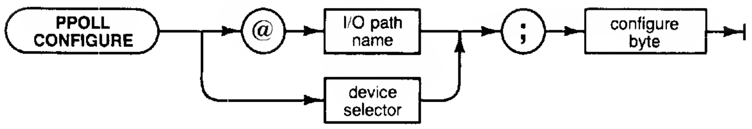
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error	ATN & EOI (duration≥25μs) Read byte $\overline{\text{EOI}}$ Restore ATN to previous state	Error
Not Active Controller	Error			

PPOLL CONFIGURE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement programs the logical sense and data bus line on which a specified device responds to a parallel poll.



Item	Description/Default	Range Restrictions	Recommended Range
I/O path name	name assigned to a device or devices	any valid name	—
device selector	numeric expression, rounded to an integer	must contain a primary address (see Glossary)	—
configure byte	numeric expression, rounded to an integer	−32 768 thru +32 767	0 thru 15

Example Statements

```
PPOLL CONFIGURE 711;2
PPOLL CONFIGURE @Dvm;Response
```

Semantics

This statement assumes that the device's response is bus-programmable. The computer must be the active controller to execute this statement.

The configure byte is coded. The three least significant bits determine the data bus line for the response. The fourth bit determines the logical sense of the response.

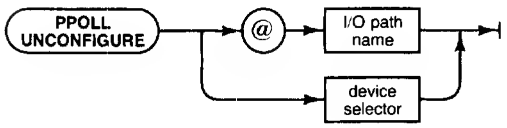
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN MTA UNL LAG PPC PPE	Error	ATN MTA UNL LAG PPC PPE
Not Active Controller	Error			

PPOLL UNCONFIGURE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement disables the parallel poll response of a specified device or devices.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
PPOLL UNCONFIGURE 7
PPOLL UNCONFIGURE @Plotter
```

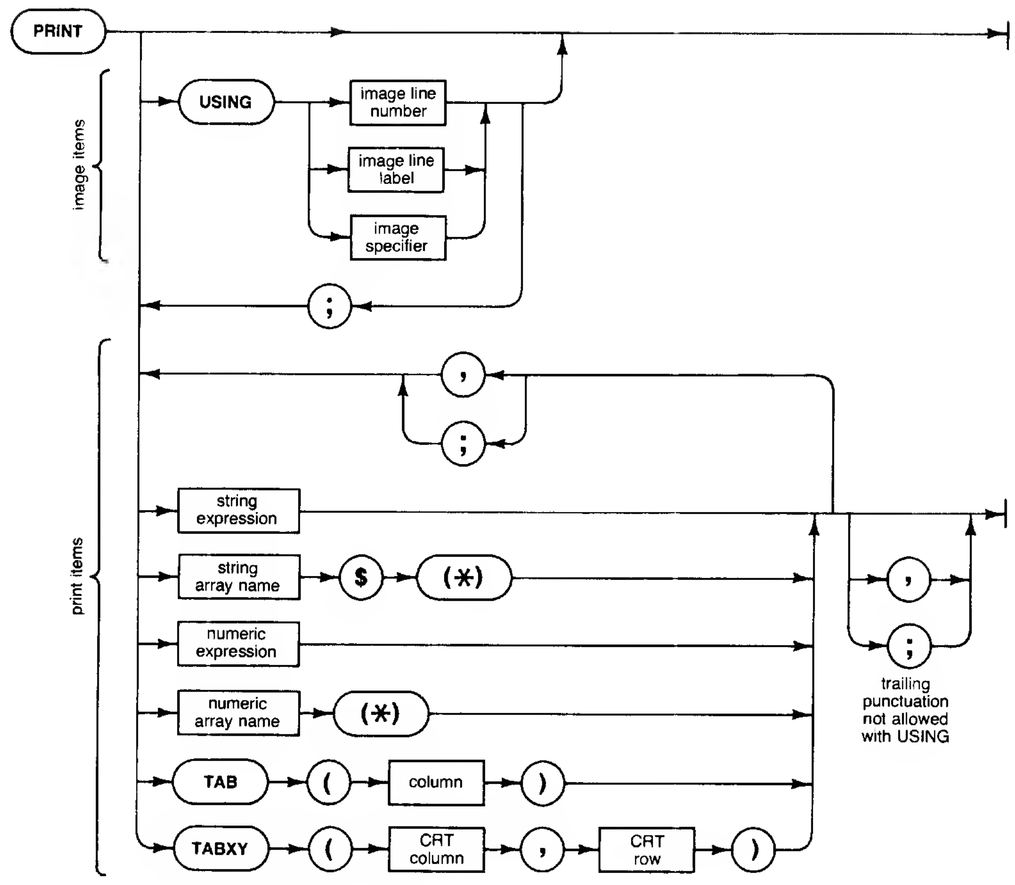
Semantics

The computer must be the active controller to execute PPOLL UNCONFIGURE.

If multiple devices are specified by an I/O path name, all specified devices are deactivated from parallel poll response. If the device selector or I/O path name refers only to an interface select code, all devices on that interface are deactivated from parallel poll response.

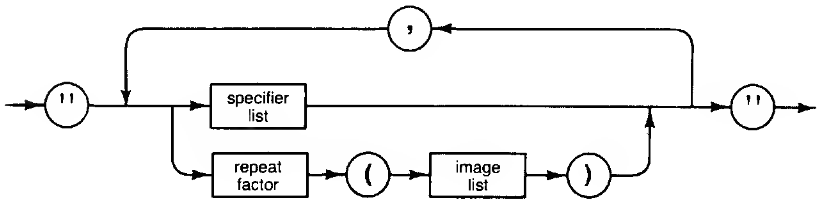
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN PPU	ATN MTA UNL LAG PPC PPD	ATN PPU	ATN MTA UNL LAG PPC PPD
Not Active Controller	Error			



tab functions not allowed with USING

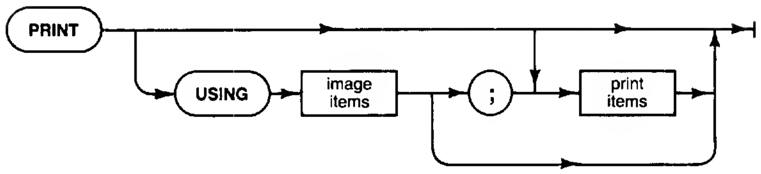
literal form of image specifier:



PRINT

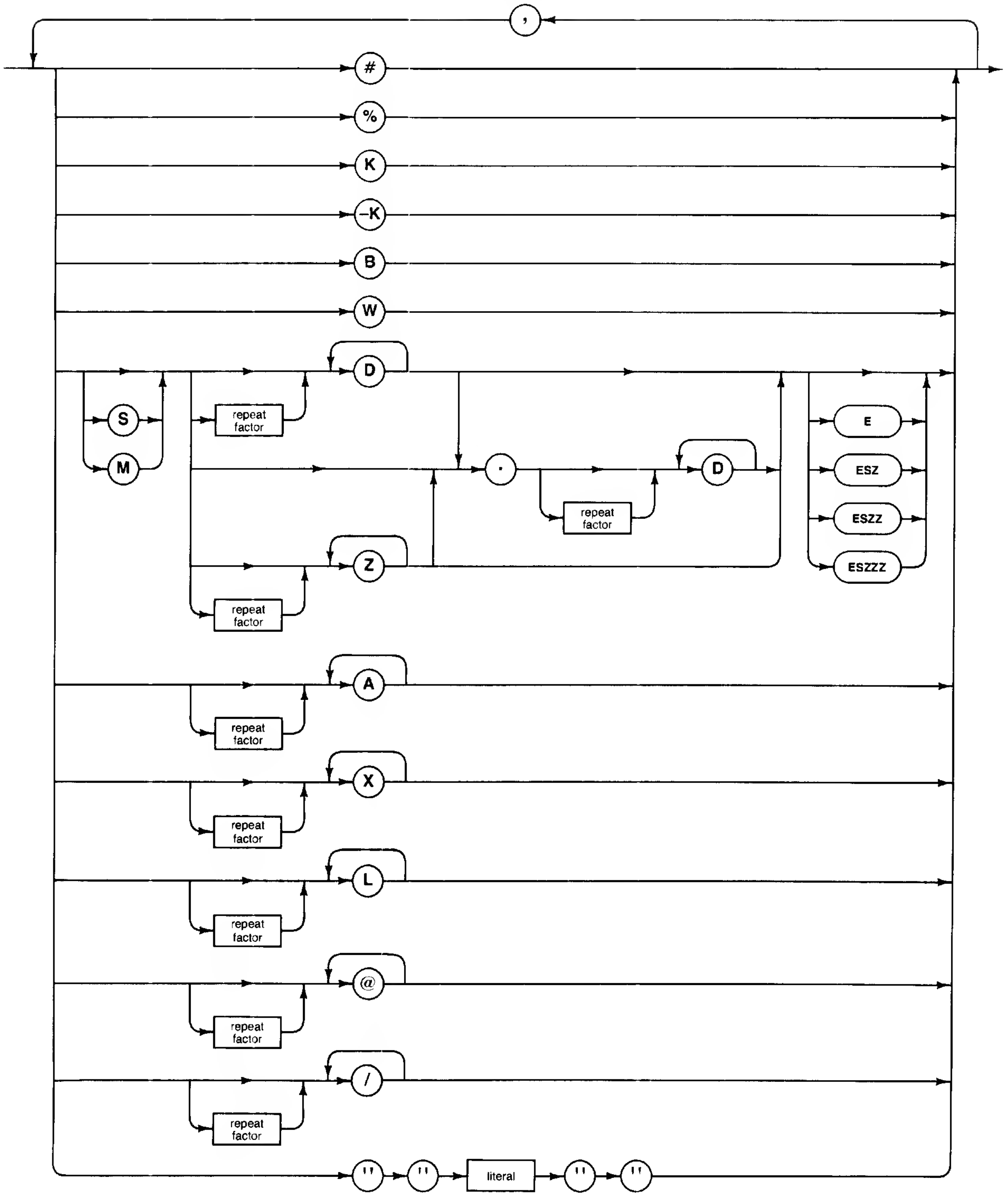
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends items to the PRINTER IS device.



Item	Description/Default	Range Restrictions	Recommended Range
image line label	name of a program line	any valid name	—
image line number	integer constant identifying a program line	1 thru 32 766	—
image specifier	string expression	(see drawing)	—
string array name	name of a string array	any valid name	—
numeric array name	name of a numeric array	any valid name	—
column	numeric expression, rounded to an integer	−32 768 thru +32 767	device dependent
CRT column	numeric expression, rounded to an integer	0 thru 32 767	1 thru 50
CRT row	numeric expression, rounded to an integer	0 thru 32 767	1 thru 18
image specifier list	literal	(see next drawing)	—
repeat factor	integer constant	1 thru 32 767	—
literal	string constant composed of characters from the keyboard, including those generated using the ANY CHAR key	quote mark not allowed	—

image specifier list



Example Statements

```
PRINT "LINE";Number
PRINT Array(*);
PRINT TABXY(1,1),Header$,TABXY(Col,3),Message$
PRINT USING "5Z.DD";Money
PRINT USING Fmt3;Id,Item$,Kilograms/2.2
```

Semantics

Standard Numeric Format

The standard numeric format depends on the value of the number being displayed. If the absolute value of the number is greater than or equal to 1E−4 and less than 1E+6, it is rounded to 12 digits and displayed in floating point notation. If it is not within these limits, it is displayed in scientific notation. The standard numeric format is used unless USING is selected, and may be specified by using K in an image specifier.

Automatic End-Of-Line Sequence

After the print list is exhausted, an End-Of-Line (EOL) sequence is sent to the PRINTER IS device, unless it is suppressed by trailing punctuation or a pound-sign (#) image specifier. The EOL sequence is also sent after every fifty characters if the CRT is the PRINTER IS device, and after every eighty characters for an external printer. This "printer width exceeded" EOL is not suppressed by trailing punctuation, but can be suppressed by the use of an image specifier.

Control Codes

Some ASCII control codes have a special effect in PRINT statements if the PRINTER IS device is the CRT (device selector = 1):

Character	Keystroke	Name	Action
CHR$(7)	CTRL-G	bell	Sounds the beeper
CHR$(8)	CTRL-H	backspace	Moves the print position back one character.
CHR$(10)	CTRL-J	line-feed	Moves the cursor down one line.
CHR$(12)	CTRL-L	form-feed	Prints two line-feeds, then advances the CRT buffer enough lines to place the next item at the top of the CRT.
CHR$(13)	CTRL-M	carriage-return	Moves the print position to column 1.

The effect of ASCII control codes on a printer is device dependent. See your printer manual to find which control codes are recognized by your printer and their effects.

Arrays

Entire arrays may be printed using the asterisk specifier. Each element in an array is treated as a separate item, as if the elements were all listed and separated by the punctuation following the array specifier. If no punctuation follows the array specifier, a comma is assumed. The array is printed in row-major order (right-most subscript varies fastest).

PRINT Fields

If PRINT is used without USING, the punctuation following an item determines the width of the item's print field; a semicolon selects the compact field, and a comma selects the default print field. Any trailing punctation will suppress the automatic EOL sequence, in addition to selecting the print field to be used for the print item preceding it.

The compact field is slightly different for numeric and string items. Numeric items are printed with one trailing blank. String items are printed with no leading or trailing blanks.

The default print field prints items with trailing blanks to fill to the beginning of the next 10-character field.

Numeric data is printed with one leading blank if the number is positive, or with a minus sign if the number is negative, whether in compact or default field.

TAB

The TAB function is used to position the next character to be printed on a line. In the TAB function, a column parameter less than one is treated as one. A column parameter greater than zero is subjected to the following formula: TAB position = ((column − 1) MOD width) + 1; where "width" is 50 for the CRT and 80 for an external printer. If the TAB position evaluates to a column number less than or equal to the number of characters printed since the last EOL sequence, then an EOL sequence is printed, followed by (TAB position − 1) blanks. If the TAB position evaluates to a column number greater than the number of characters printed since the last EOL, sufficient blanks are printed to move to the TAB position.

TABXY
The TABXY function provides X-Y character positioning on the CRT. It is ignored if a device other than the CRT is the PRINTER IS device. TABXY(1,1) specifies the upper left-hand corner of the CRT. If a negative value is provided for CRT row or CRT column, it is an error. Any number greater than 50 for CRT column is treated as 50. Any number greater than 18 for CRT row is treated as 18. If 0 is provided for either parameter, the current value of that parameter remains unchanged.

PRINT With Using
When the computer executes a PRINT USING statement, it reads the image specifier, acting on each field specifier (field specifiers are separated from each other by commas) as it is encountered. If nothing is required from the print items, the field specifier is acted upon without accessing the print list. When the field specifer requires characters, it accesses the next item in the print list, using the entire item. Each element in an array is considered a separate item.

The processing of image specifiers stops when a specifier is encountered that has no matching print item. If the image specifiers are exhausted before the print items, they are reused, starting at the beginning.

If a numeric item requires more decimal places to the left of the decimal point than are provided by the field specifier, an error is generated. A minus sign takes a digit place if M or S is not used, and can generate unexpected overflows of the image field. If the number contains more digits to the right of the decimal point than are specified, it is rounded to fit the specifier.

If a string is longer than the field specifier, it is truncated, and the rightmost characters are lost. If it is shorter than the specifer, trailing blanks are used to fill out the field.

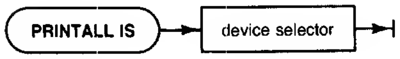
Effects of the image specifiers on a PRINT statement are shown in the following table.

Image Specifier	Meaning
K −K	Compact field. Prints a number or string in standard form −K with no leading or trailing blanks.
S	Prints the number's sign (+ or −).
M	Prints the number's sign if negative, a blank if positive.
D	Prints one digit character. A leading zero is replaced by a blank. If the number is negative and no sign image is specified, the minus sign will occupy a leading digit position. If a sign is printed, it will "float" to the left of the left-most digit.
Z	Same as D, except that leading zeros are printed.
B	Prints the character represented by one byte of data. This is similar to the CHR$ function. The least significant eight bits of the number are sent. The number is rounded to an integer. If the number is greater than 32 767, 255 is used; if the number is less than −32 768, 0 is used.
W	Prints two characters represented by the two bytes in a 16-bit word. The number is rounded to an integer. If the number is larger than 32 767, 32 767 is used; if the number is less than −32 768, then −32 768 is used. On an 8-bit interface, the most significant byte is sent first, followed by the least significant byte.
A	Prints a string character. Trailing blanks are output if the number of characters specified is greater than the number available in the corresponding string. If the image specifier is exhausted before the corresponding string, the remaining characters are ignored.
X	Prints a blank.
.	Prints a decimal point radix indicator.
E ESZZ	Prints an E, a sign, and a two digit exponent.
ESZ	Prints an E, a sign, and a one digit exponent.
ESZZZ	Prints an E, a sign, and a three digit exponent.
#	Suppresses all automatic output of the EOL (End-Of-Line) sequence.
%	Ignored in PRINT images.
L	Sends an EOL sequence to the PRINTER IS device.
@	Sends a form-feed to the PRINTER IS device.
/	Sends a carriage-return and a line-feed to the PRINTER IS device.
literal	Prints the characters contained in the literal.

PRINTALL IS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement assigns a logging device for recording operator interaction and troubleshooting messages.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
PRINTALL IS 701
PRINTALL IS Gpio
```

Semantics

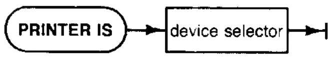
The printall device must be enabled by the PRT ALL key on the computer. The PRT ALL key is a toggle action device, enabling and disabling the printall operation. When the printall mode is enabled, all items generated by DISP, all operator input followed by the ENTER, CONTINUE, or EXECUTE key, and all error messages from the computer are logged on the printall device. All TRACE activity is logged on the printall device if tracing is enabled.

At power-on and SCRATCH A, the printall device is the CRT (device selector = 1).

PRINTER IS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the system printing device for all PRINT, CAT and LIST statements which do not specify a destination. The PRINTER IS device is 1 (the CRT) at power-on and after SCRATCH A.



Item	Description/Default	Range Restrictions
device selector	numeric expression, rounded to an integer	(see Glossary)

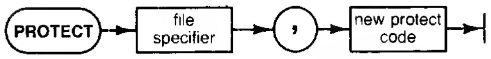
Example Statements

```
PRINTER IS 701
PRINTER IS Gpio
```

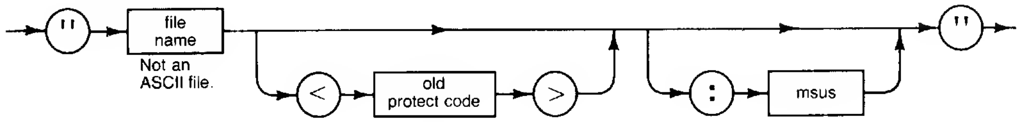
PROTECT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement specifies the protect code used on PROG, BDAT, and BIN files.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
old protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL
new protect code	string expression; first two characters are significant	">" not allowed

Example Statements

```
PROTECT Name$,Pc$
PROTECT "George<xy>:INTERNAL","NEW"
```

Semantics

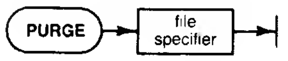
A protect code guards against accidental changes to an individual file. Once a file is protected, the protect code must be included in its file specifier for all operations except LOAD, LOADBIN, and LOADSUB.

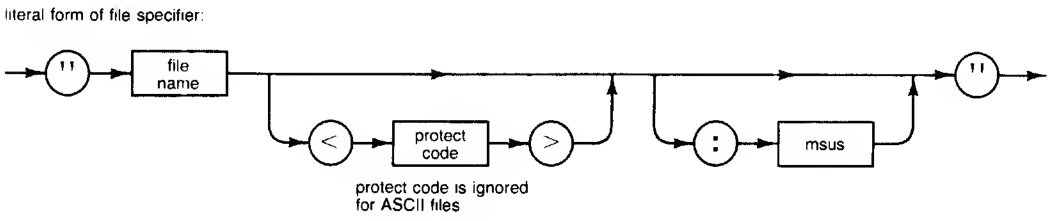
Removing a protect code from a file is accomplished by assigning a protect code that contains blanks for the first two characters of the string expression or is the null string.

PURGE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement deletes a file entry from the directory of the mass storage media.




literal form of file specifier:
"
file name
<
protect code
>
protect code is ignored for ASCII files
:
msus
"

Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

Example Statements

```
PURGE Name$
PURGE "George<PC>"
```

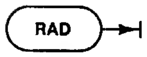
Semantics

Once a file is purged, you cannot access the information which was in the file. The records of a purged file are returned to "available space". An open file must be closed before it can be purged. Any file can be closed by ASSIGN...TO * (see ASSIGN).

RAD

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects radians as the unit of measure for expressing angles.



Semantics

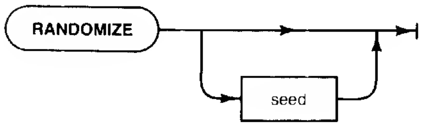
All functions which return an angle will return an angle in radians. All operations with parameters representing angles will interpret the angle in radians. If no angle mode is specified in a program, the default is radians (also see DEG).

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

RANDOMIZE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement selects a seed for the RND function.



Item	Description/Default	Range Restrictions	Recommended Range
seed	numeric expression, rounded to an integer; Default = pseudorandom	—	1 thru $2^{31}-2$

Example Statements

```
RANDOMIZE
RANDOMIZE Old_seed*PI
```

Semantics

The seed actually used by the random number generator depends on the absolute value of the seed specified in the RANDOMIZE statement.

Absolute Value of Seed	Value Used
less than 1	1
1 thru $2^{31}-2$	INT(ABS(seed))
greater than $2^{31}-2$	$2^{31}-2$

The seed is reset to 37 480 660 by power-up, SCRATCH A, SCRATCH, and program prerun.

RATIO

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the ratio of the X hard clip limits to the Y hard clip limits for the current PLOTTER IS device.

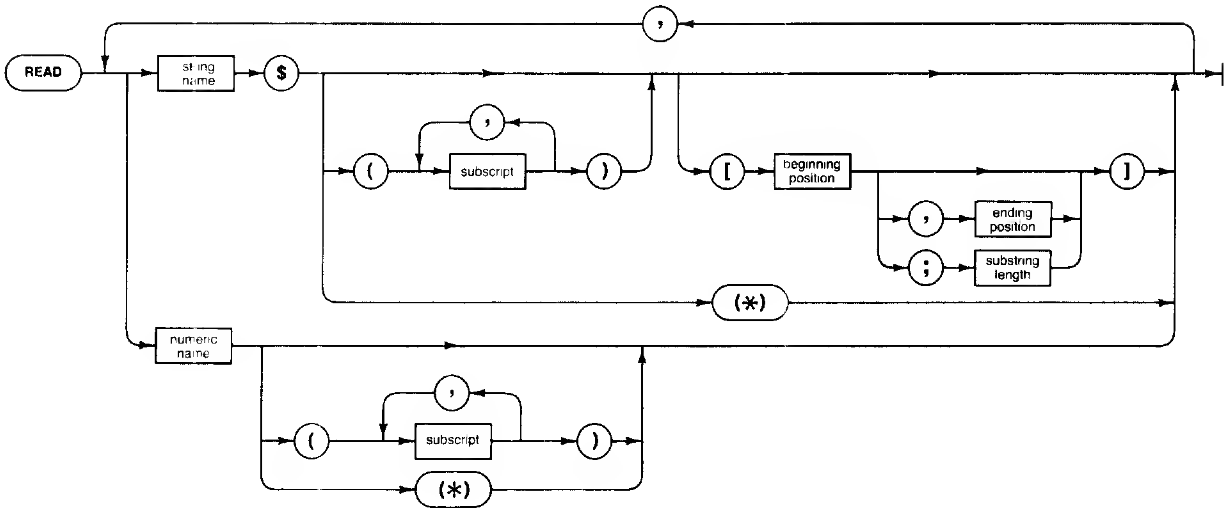
Example Statements

```
WINDOW 0,10*RATIO,-10,10
Turn=1/RATIO
```

READ

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement reads values from DATA statements and assigns them to variables.



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	−32 767 thru +32 767 (see "array" in Glossary)
beginning position	numeric expression, rounded to an integer	1 thru 32 767 (see "substring" in Glossary)
ending position	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)
substring length	numeric expression, rounded to an integer	0 thru 32 767 (see "substring" in Glossary)

Example Statements

```
READ Number,String$
READ Array(*)
READ Item(1,1),Item(2,1),Item(3,1)
```

Semantics

The numeric items stored in DATA statements are considered strings by the computer, and are processed with a VAL function to be read into numeric variables in a READ statement. If they are not of the correct form, error 32 may result. Real DATA items will be rounded into an INTEGER variable if they are within the INTEGER range (−32 768 thru 32 767). A string variable may read numeric items, as long as it is dimensioned large enough to contain the characters.

The first READ statement in a context accesses the first item in the first DATA statement in the context unless RESTORE has been used to specify a different DATA statement as the starting point. Successive READ operations access following items, progressing through DATA statements as necessary. Trying to READ past the end of the last DATA statement results in error 36. The order of accessing DATA statements may be altered by using the RESTORE statement.

An entire array can be specified by replacing the subscript list with an asterisk. The array entries are made in row major order (right most subscript varies most rapidly).

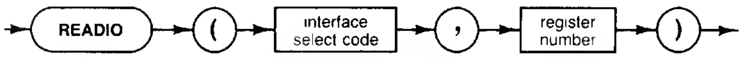
READIO

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function reads the contents of the specified hardware register on the specified interface.

Note

Unexpected results may occur with select codes outside the given range.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	1 thru 31
register number	numeric expression, rounded to an integer	interface dependent

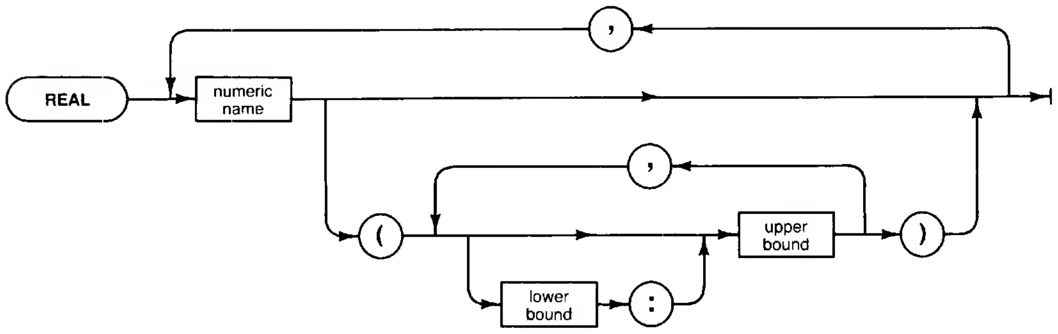
Example Statements

```
Upper_byte=READIO(Gpio,4)
PRINT "Register";I;"=";READIO(7,I)
```

REAL

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement reserves storage for floating point variables and arrays. (For information about REAL as a secondary keyword, see the ALLOCATE, COM, DEF FN, or SUB statements.)



Item	Description/Default	Range Restrictions
numeric name	name of a numeric variable	any valid name
lower bound	integer constant; Default = OPTION BASE value (0 or1)	−32 767 thru +32 767 (see "array" in Glossary)
upper bound	integer constant	−32 767 thru +32 767 (see "array" in Glossary)

Example Statements

```
REAL X,Y,Z
REAL Array(-128:127,15)
```

Semantics

Each REAL variable or array element requires eight bytes of number storage. The maximum number of subscripts in an array is six, and no dimension may have more than 32 767 elements. The total number of elements in an array is limited by memory.

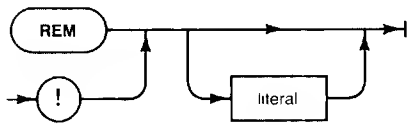
RECOVER

See the ON ERROR, ON END, ON KEY, ON KNOB, ON INTR, and ON TIMEOUT statements.

REM

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This statement allows comments in a program.



Item	Description/Default	Range Restrictions
literal	string constant composed of characters from the keyboard, including those generated with the ANY CHAR key	—

Example Program Lines

```
100  REM   Program Title
190  !
200  IF BIT(Info,2) THEN Branch  ! Test overrange bit
```

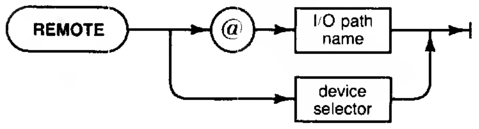
Semantics

REM must be the first keyword on a program line. If you want to add comments to a statement, an exclamation point must be used to mark the beginning of the comment. If the first character in a program line is an exclamation point, the line is treated like a REM statement and is not checked for syntax.

REMOTE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement places HP-IB devices having remote/local capabilities into the remote state.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
REMOTE 712
REMOTE @Hpib
```

Semantics

If individual devices are not specified, the remote state for all devices on the bus having remote/local capabilities is enabled. The bus configuration is unchanged, and the devices switch to remote if and when they are addressed to listen. If primary addressing is used, only the specified devices are put into the remote state.

When the computer is the system controller and is switched on, reset, or ABORT is executed, bus devices are automatically enabled for the remote state and switch to remote when they are addressed to listen.

The computer must be the system controller to execute this statement, and it must be the active controller to place individual devices in the remote state.

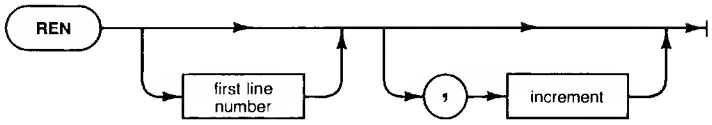
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	REN	REN ATN MTA UNL LAG	Error	
Not Active Controller	REN	Error	Error	

REN

Keyboard Executable	Yes
Programmable	No

This command renumbers the lines in a program.



Item	Description/Default	Range Restrictions
first line number	integer constant identifying a program line; Default = 10	1 thru 32 766
increment	integer constant; Default = 10	1 thru 32 765

Example Statements

```
REN 1000
REN 100,2
```

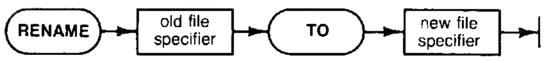
Semantics

The renumbered program will begin with the specified first line number, and subsequent lines will be separated by the increment. If a renumbered line is referenced by a statement such as GOTO or GOSUB, the reference to that line is adjusted to reflect the new line number. REN on a paused program causes it to move to the stopped state.

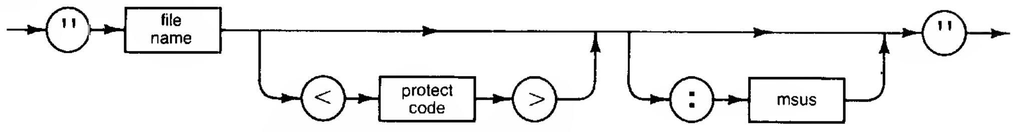
RENAME

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement changes a file's name in the mass storage media's directory.



literal form of file specifier:



protect code and msus are ignored in new file specifier

all protect codes are ignored for ASCII files

Item	Description/Default	Range Restrictions
old file specifier	string expression	(see drawing)
new file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL

Example Statements

```
RENAME "TEMP<pc>" TO "FINAL"
RENAME Name$&Msus$ TO Temp$
```

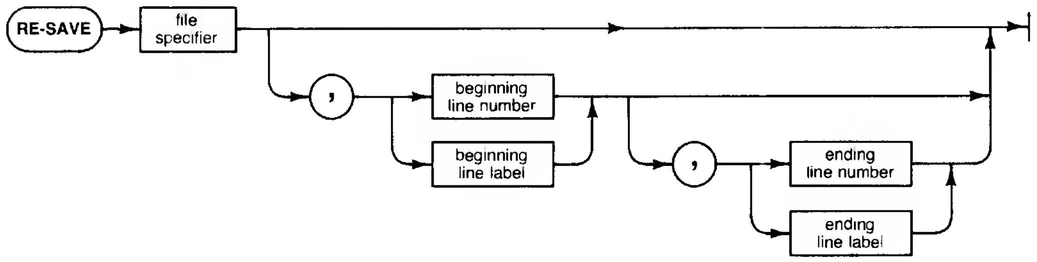
Semantics

The new file name must not duplicate the name of any other file currently in the directory. A protected file retains its old protect code, which must be included in the old file specifier.

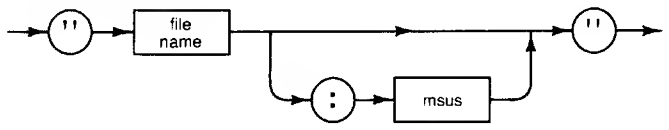
RE-SAVE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file and copies program lines as strings into that file.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
RE-SAVE "George"
RE-SAVE Name$,1,Sort
```

Semantics

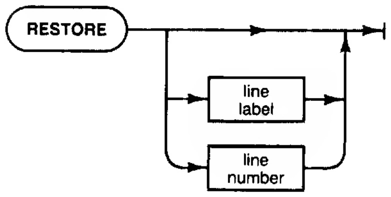
An entire program can be saved, or the portion delimited by providing beginning and (if needed) ending line labels or numbers. If the file named already exists, the old file entry is removed from the directory after the new file is successfully saved on the mass storage media. Pressing RESET during a RE-SAVE operation results in the old file being retained. Attemping to RE-SAVE any file that is not an ASCII file results in an error.

If a specified line label does not exist, error 3 occurs. If a specified line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs.

RESTORE

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

RESTORE specifies which DATA statement will be used by the next READ operation.



Item	Description/Default	Range Restrictions
line label	name of a program line	any valid name
line number	integer constant identifying a program line; Default = first DATA statement in context	1 thru 32 766

Example Statements

```
RESTORE
RESTORE Third_array
```

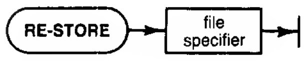
Semantics

If a line is specified which does not contain a DATA statement, the computer uses the first DATA statement after the specified line. RESTORE can only refer to lines within the current context.

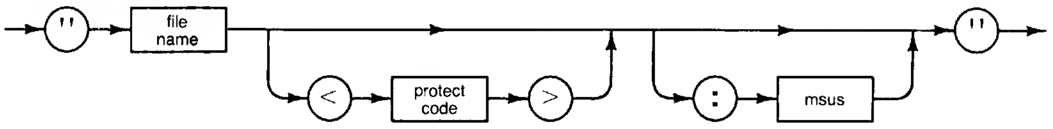
RE-STORE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a PROG file and stores an internal form of the BASIC program and all normal binary programs into that file.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL

Example Statements

```
RE-STORE "MYPROG<pc>"
RE-STORE Name$&Msus$
```

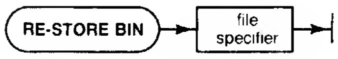
Semantics

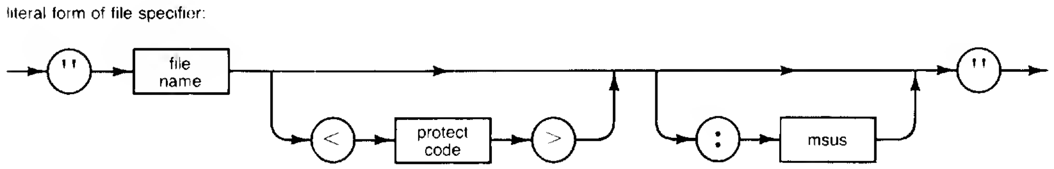
If the file named already exists, the old file entry is removed from the directory after the new file is successfully saved on the mass storage media. Pressing RESET during a RE-STORE operation results in the old file being retained. If the file being replaced had a protect code, the same protect code must be used in the RE-STORE operation. Attemping to RE-STORE any file that is not a PROG file results in an error.

RE-STORE BIN

Keyboard Executable	Yes
Programmable	No

This statement creates a BIN file and stores all normal binary programs into that file.





Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed

msus	literal;
	Default = MASS STORAGE IS device
		INTERNAL

Example Statements

```
RE-STORE BIN "BINPROG<pc>"
RE-STORE BIN Name$&Msus$
```

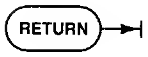
Semantics

If the file named already exists, the old file entry is removed from the directory after the new file is successfully saved on the mass storage media. Pressing RESET during a RE-STORE BIN operation results in the old file being retained. If the file being replaced had a protect code, the same protect code must be used in the RE-STORE BIN operation.

RETURN

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement returns program execution to the line following the invoking GOSUB. The keyword RETURN is also used in user-defined functions (see DEF FN).



RND

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a pseudo-random number greater than 0 and less than 1.

Example Statements

```
Percent=RND*100
IF RND<.5 THEN Case1
```

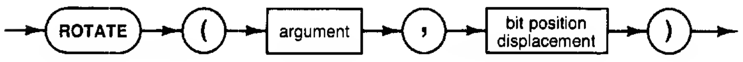
Semantics

The random number returned is based on a seed set to 37 480 660 at power-on, SCRATCH, SCRATCH A, or program prerun. Each succeeding use of RND returns a random number which uses the previous random number as a seed. The seed can be modified with the RANDOMIZE statement.

ROTATE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified. The shift is performed with wraparound.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	—
bit position displacement	numeric expression, rounded to an integer	−32 768 thru +32 767	−15 thru +15

Example Statements

```
New_word=ROTATE(Old_word,2)
Q=ROTATE(Q,Places)
```

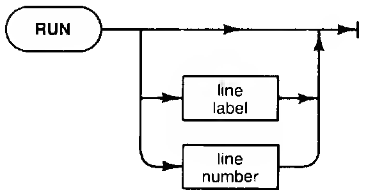
Semantics

The argument is converted into a 16-bit, two's-complement form. If the bit position displacement is positive, the rotation is towards the least-significant bit. If the bit position displacement is negative, the rotation is towards the most-significant bit. The rotation is performed without changing the value of any variable in the argument.

RUN

Keyboard Executable	Yes
Programmable	No

This command starts program execution at a specified line.



Item	Description/Default	Range Restrictions
line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
line label	name of a program line	any valid name

Example Statements

```
RUN 10
RUN Part2
```

Semantics

Pressing the RUN key is the same as executing RUN with no label or line number. RUN is executed in two phases: prerun initialization and program execution.

The prerun phase consists of:

- Reserving memory space for variables specified in COM statements (both labeled and blank).
- Reserving memory space for variables specified by DIM, REAL, INTEGER, or implied in the main program segment. This does not include variables used with ALLOCATE, which is done at run-time.
- Checking for syntax errors which require more than one program line to detect. Included in this are errors such as incorrect array references, and mismatched parameter or COM lists.

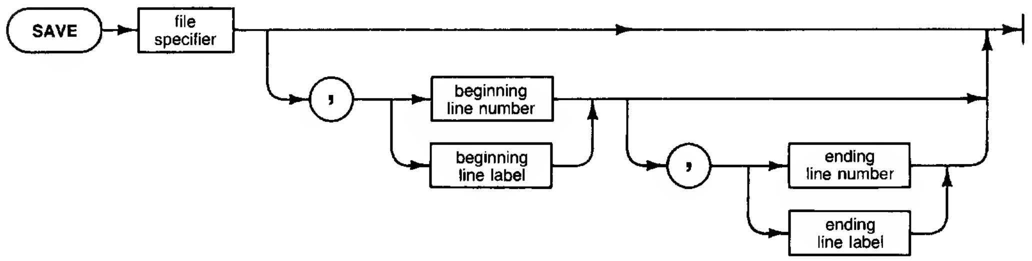
If an error is detected during prerun phase, prerun halts and an error message is displayed on the CRT.

After successful completion of prerun initialization, program execution begins with either the lowest numbered program line or the line specified in the RUN command. If the line number specified does not exist in the main program, execution begins at the next higher-numbered line. An error results if there is no higher-numbered line available within the main program, or if the specified line label cannot be found in the main program.

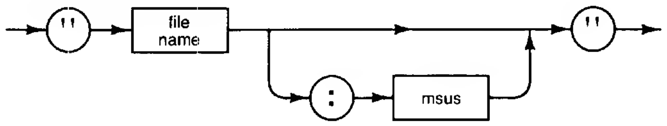
SAVE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates an ASCII file and copies program lines as strings into that file.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
msus	literal; Default = MASS STORAGE IS device	INTERNAL
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
SAVE "WHALES"
SAVE "TEMP",1,Sort
```

Semantics

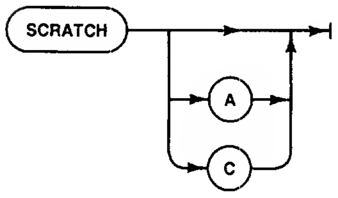
An entire program can be saved, or any portion delimited by the beginning and (if needed) ending line numbers or labels. This statement is for creating new files. Attempting to SAVE a file name that already exists causes error 54. If you need to replace an old file, see RE-SAVE.

If a specified line label does not exist, error 3 occurs. If a specified line number does not exist, the program lines with numbers inside the range specified are saved. If the ending line number is less than the beginning line number, error 41 occurs.

SCRATCH

Keyboard Executable	Yes
Programmable	No

This command erases all or selected portions of memory. See the Reset Table in the back of this book for details on the effects of SCRATCH.



Example Statements

```
SCRATCH
SCRATCH A
```

Semantics

SCRATCH clears the BASIC program from memory. All variables not in COM are also cleared.

SCRATCH C clears all variables, including those in COM. The program is left intact.

SCRATCH A clears the BASIC program memory, all normal binary programs, and all variables, including those in COM. Most internal parameters in the computer are reset by this command.

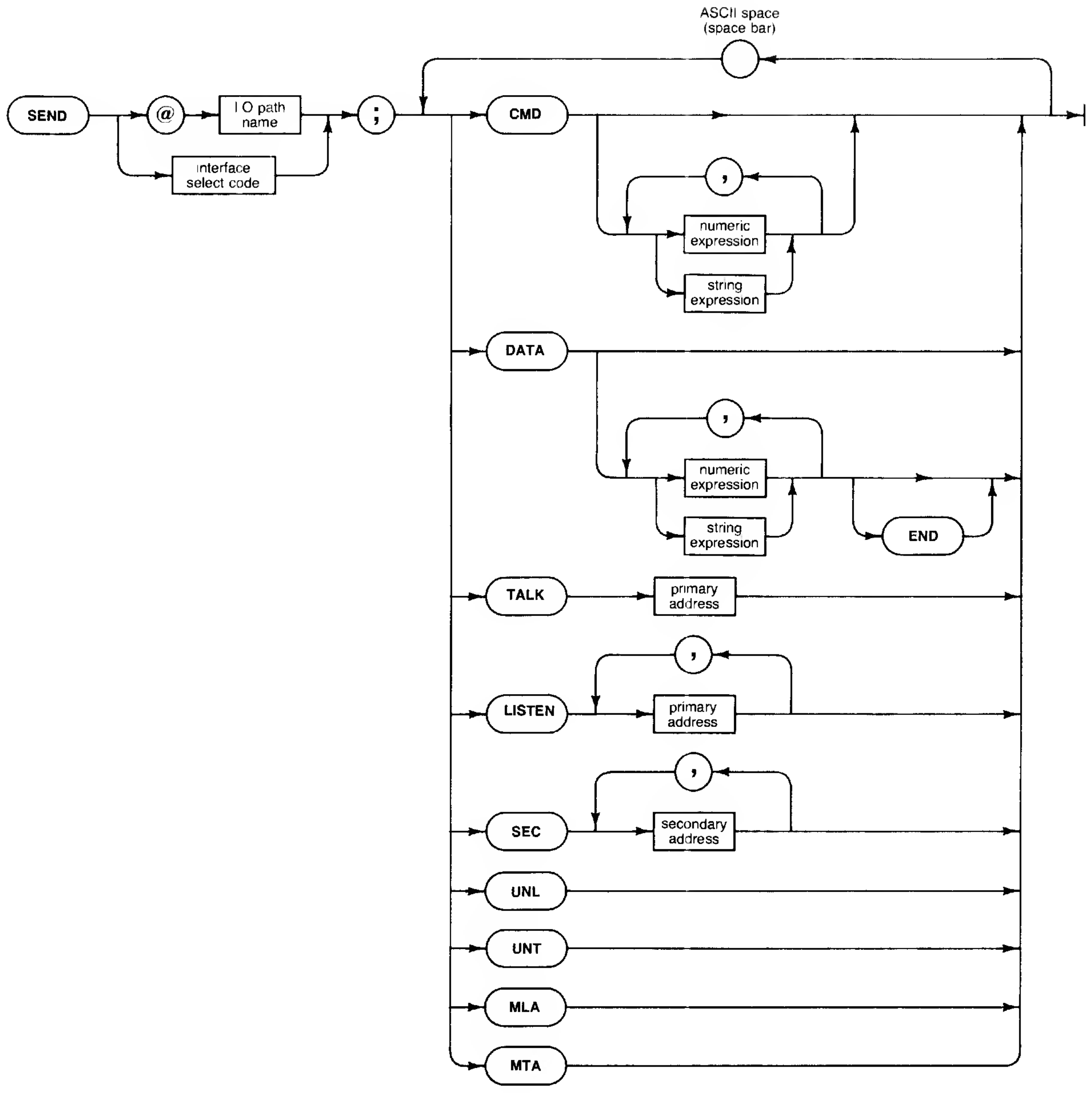
SEC

See the SEND statement.

SEND

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends messages to an HP-IB.



Item	Description/Default	Range Restrictions
interface select code	numeric expression, rounded to an integer	7 thru 31
I/O path name	name assigned to an interface select code	any valid name (see ASSIGN)
primary address	numeric expression, rounded to an integer	0 thru 31
secondary address	numeric expression, rounded to an integer	0 thru 31

Example Statements

```
SEND 7;UNL MTA LISTEN 1 DATA "HELLO" END
SEND @Hpib;UNL MLA TALK Device CMD 24+128
```

Semantics

CMD

The expressions following a CMD are sent with ATN true. The ASCII characters representing the evaluated string expression are sent to the HP-IB. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. CMD with no items sets ATN true.

DATA

The expressions following DATA are sent with ATN false. The ASCII characters representing the evaluated string expression are sent. Numeric expressions are rounded to an integer MOD 256. The resulting byte is sent to the HP-IB. If END is added to the data list, EOI is set true before sending the last byte. DATA with no items sets ATN false without waiting to be addressed as a talker.

If the computer is active controller, and addressed as a talker, the data is sent immediately. If the computer is not active controller, it waits until it is addressed to talk before sending the data.

TALK

TALK sets ATN true and sends the specified talk address. Only one primary address is allowed for a single talker. An extended talker may be addressed by using SEC secondary address after TALK. A TALK address of 31 is equivalent to UNT (untalk).

UNT

UNT sets ATN true and sends the untalk command. (There is no automatic untalk.) A TALK address of 31 is equivalent to UNT.

LISTEN

LISTEN sets ATN true, sends one or more primary addresses, and addresses those devices to listen. A LISTEN address of 31 is equivalent to UNL (unlisten).

UNL

UNL set ATN true and sends the unlisten command. (There is no automatic unlisten.) A LISTEN address of 31 is equivalent to UNL.

SEC
SEC sets ATN true and sends one or more secondary addresses (commands).

MTA
MTA sets ATN true and sends the interface's talk address. It is equivalent to performing a status sequence on the interface and then using the returned talk address with a SEND..TALK sequence.

MLA
MLA sets ATN true and sends the interface's listen address. It is equivalent to performing a status sequence on the interface and then using the returned listen address with a SEND..LISTEN sequence.

Summary
The computer must be the active controller to execute SEND with CMD, TALK, UNT, LISTEN, UNL, SEC, MTA and MLA.

The computer does not have to be the active controller to send DATA. DATA is sent when the computer is addressed to talk.

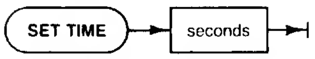
The following table lists the HP-IB message mnemonics, descriptions of the messages, and the secondary keywords required to send the messages. Any numeric values are decimal.

Mnemonic	Description	Secondary Keyword and Value
DAB	Data Byte	DATA 0 thru DATA 255
DCL	Device Clear	CMD 20 or CMD 148
EOI	End or Identify	DATA (data) END
GET	Group Execute Trigger	CMD 8 or CMD 136
GTL	Go To Local	CMD 1 or CMD 129
IFC	Interface Clear	Not possible with SEND. An ABORT statement must be used.
LAG	Listen Address Group	LISTEN 0 thru LISTEN 31 or CMD 32 thru CMD 63
MLA	My Listen Address	MLA
MTA	My Talk Address	MTA
PPC	Parallel Poll Configure	CMD 5 or CMD 133
PPD	Parallel Poll Disable	PPC (CMD 5 or CMD 133), followed by CMD 112, or CMD 240, or SEC 16.
PPE	Parallel Poll Enable	PPC (CMD 5 or CMD 133), followed by CMD 96 thru CMD 111, or CMD 224 thru CMD 239, or SEC 0 thru SEC 15. SEC 0 allows a mask to be specified by a numeric value.
PPU	Parallel Poll Unconfigure	CMD 21 or CMD 149
PPOLL	Parallel Poll	Not possible with SEND. PPOLL function must be used.
REN	Remote Enable	Not possible with SEND. REMOTE statement must be used.
SDC	Selected Device Clear	CMD 4 or CMD 132
SPD	Serial Poll Disable	CMD 25 or CMD 153
SPE	Serial Poll Enable	CMD 24 or CMD 152
TAD	Talk Address	TALK 0 thru TALK 31, or CMD 64 thru CMD 95, or CMD 192 thru CMD 223.
TCT	Take Control	CMD 9 or CMD 137
UNL	Unlisten	UNL, or LISTEN 31, or CMD 63, or CMD 191.
UNT	Untalk	UNT, or TALK 31, or CMD 95, or CMD 223.

SET TIME

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement resets the time-of-day given by the real-time clock.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest hundredth	0 thru 86 399.99

Example Statements

```
SET TIME 0
SET TIME Hours*3600+Minutes*60
```

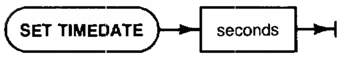
Semantics

SET TIME changes only the time within the current day, not the date. The new clock setting is equivalent to (TIMEDATE DIV 86 400) × 86 400 plus the specified setting.

SET TIMEDATE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement resets the absolute seconds (time and day) given by the real-time clock.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest hundredth	2.086 629 12 E+11 thru 2.143 252 223 999 9 E+11

Example Statements

```
SET TIMEDATE TIMEDATE+86400
SET TIMEDATE Strange_number
```

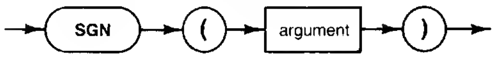
Semantics

The clock is set to 2.086 629 12 E+11 (midnight March 1, 1900) at power-on. The clock values represent Julian time, expressed in seconds.

SGN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns 1 if the argument is positive, 0 if it equals zero, and −1 if it is negative.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

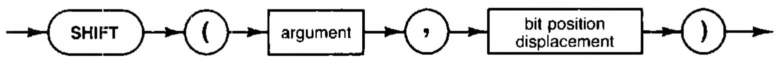
Example Statements

```
Root=SGN(X)*SQR(ABS(X))
Z=2*PI*SGN(Y)
```

SHIFT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer which equals the value obtained by shifting the 16-bit binary representation of the argument the number of bit positions specified, without wraparound.



Item	Description/Default	Range Restrictions	Recommended Range
argument	numeric expression, rounded to an integer	−32 768 thru +32 767	—
bit position displacement	numeric expression, rounded to an integer	−32 768 thru +32 767	−15 thru +15

Example Statements

```
New_word=SHIFT(Old_word,-2)
Mask=SHIFT(1,Position)
```

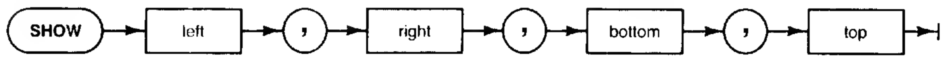
Semantics

If the bit position displacement is positive, the shift is towards the least-significant bit. If the bit position displacement is negative, the shift is towards the most-significant bit. Bits shifted out are lost. Bits shifted in are zeros. The SHIFT operation is performed without changing the value of any variable in the argument.

SHOW

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to define an isotropic current unit-of-measure for graphics operations.



Item	Description/Default	Range Restrictions
left	numeric expression	—
right	numeric expression	—
bottom	numeric expression	—
top	numeric expression	—

Example Statements

```
SHOW -5,5,0,100
SHOW Left,Right,Bottom,Top
```

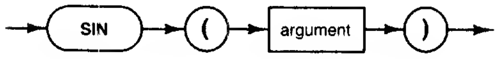
Semantics

SHOW defines the values which must be displayed within the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. SHOW creates isotropic units (units the same in X and Y). The direction of an axis may be reversed by specifying the left greater than the right or the bottom greater than the top. (Also see WINDOW.)

SIN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the sine of the angle represented by the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 1.708 312 781 2 E+10 deg. or 2.981 568 26 E+8 rad.

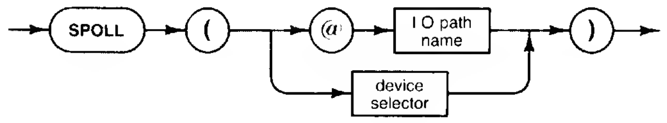
Example Statements

```
Sine=SIN(Angle)
PRINT "Sine of";Theta;"=";SIN(Theta)
```

SPOLL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns an integer containing the serial poll response from the addressed device.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	must include a primary address (see Glossary)

Example Statements

```
Stat=SPOLL(707)
IF SPOLL(@Device) THEN Respond
```

Semantics

The computer must be the active controller to execute this statement. Multiple listeners are not allowed. One secondary address may be specified to get status from an extended talker. Refer to the documentation provided with the device being polled for information concerning the device's status byte.

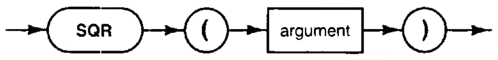
Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT	Error	ATN UNL MLA TAD SPE $\overline{\text{ATN}}$ Read data ATN SPD UNT
Not Active Controller	Error			

SQR

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the square root of the argument.



Item	Description/Default	Range Restrictions
argument	numeric expression	$\geqslant 0$

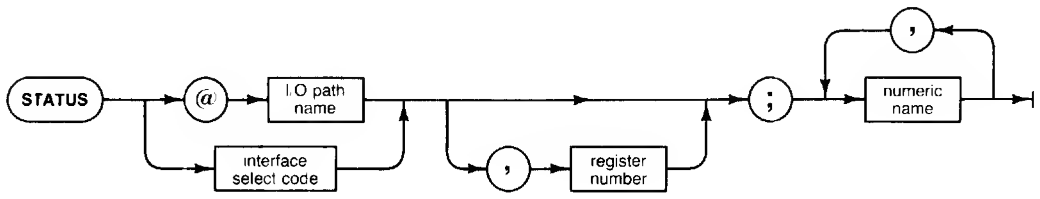
Example Statements

```
Amps=SQR(Watts/Ohms)
PRINT "Square root of";X;"=";SQR(X)
```

STATUS

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement returns the contents of interface or I/O path name status registers.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
interface select code	numeric expression, rounded to an integer	1 thru 31
register number	numeric expression, rounded to an integer; Default = 0	interface dependent
numeric name	name of a numeric variable	any valid name

Example Statements

```
STATUS 1;Xpos,Ypos
STATUS @File,5;Record
```

Semantics

The value of the beginning register number is copied into the first variable, the next register value into the second variable, and so on. The information is read until the variables in the list are exhausted, there is no wraparound to the first register and an attempt to read a nonexistent register generates an error.

The register meanings depend on the item currently associated with the I/O path name, or the specified interface. Refer to the Interface Registers section to determine the register meanings.

STEP

See the FOR...NEXT statement.

STOP

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement terminates execution of the program.

Semantics

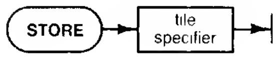
Once a program is stopped, it cannot be resumed by CONTINUE. RUN must be executed to restart the program. PAUSE should be used if you intend to continue execution of the program.

A program can have multiple STOP statements. Encountering an END statements or pressing the (STOP) key has the same effect as executing STOP. After a STOP, variables that existed in the main context are available from the keyboard.

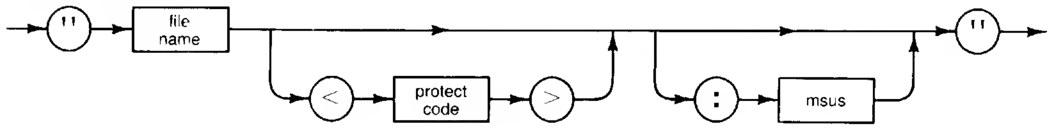
STORE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement creates a PROG file and stores an internal form of the BASIC program and all normal binary programs into the file.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL

Example Statements

```
STORE "PROG2<pc>"
STORE Name$&Msus$
```

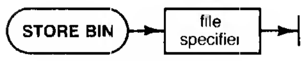
Semantics

This statement creates a new file; to replace an old file, see RE-STORE. If a protect code is specified, it becomes the protect code of the new file.

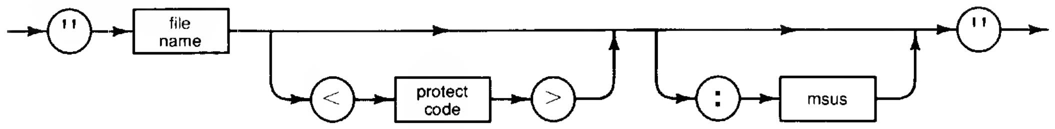
STORE BIN

Keyboard Executable	Yes
Programmable	No

This command creates a BIN file and stores all normal binary programs into the file.



literal form of file specifier:



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; first two characters are significant	">" not allowed
msus	literal; Default = MASS STORAGE IS device	INTERNAL

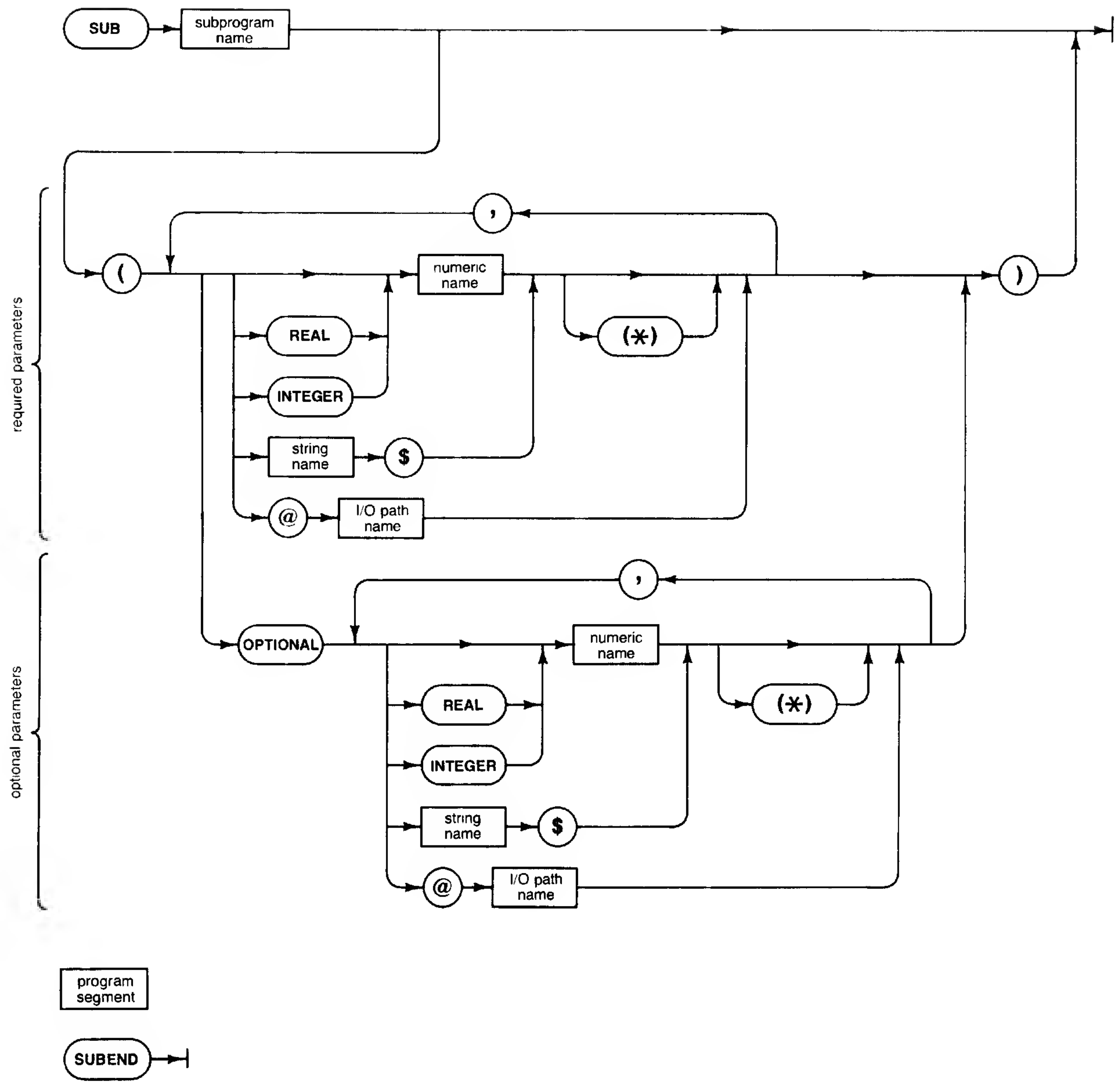
Example Statements

```
STORE BIN "FFT"
STORE BIN "NAME<pc>:INTERNAL"
```

Semantics

This statement is for creating a new file, to replace an old file, see RE-STORE BIN. If a protect code is specified, it becomes the protect code of the new file.

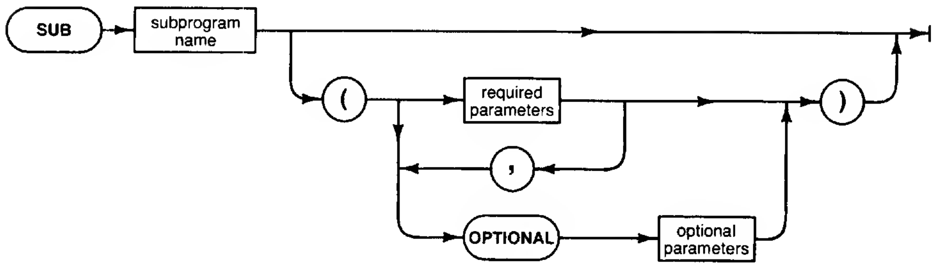
STORE BIN is not allowed while a program is running. If it is executed while the program is paused, the program moves to the stopped state.


SUB
subprogram name
required parameters
(
,
numeric name
REAL
INTEGER
string name
$
@
I/O path name
(*)
)
optional parameters
OPTIONAL
,
numeric name
REAL
INTEGER
string name
$
@
I/O path name
(*)
program segment
SUBEND

SUB

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This is the first statement in a SUB subprogram and can specify the subprogram's formal parameters.



Item	Description/Default	Range Restrictions
subprogram name	name of the SUB subprogram	any valid name
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
I/O path name	name assigned to a device, devices, or mass storage file	any valid name (see ASSIGN)
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram	—

Example Statements

```
SUB Parse(String$)
SUB Transform(@Printer,INTEGER Array(*),OPTIONAL Text$)
```

Semantics

SUB subprograms must appear after the main program. The first line of the subprogram must be a SUB statement. The last line must be a SUBEND statement. Comments after the SUBEND are considered to be part of the subprogram.

Parameters to the left of the keyword OPTIONAL are required and must be supplied whenever the subprogram is invoked (see CALL). Parameters to the right of OPTIONAL are optional, and only need to be supplied if they are needed for a specific operation. Optional parameters are associated from left to right with any remaining pass parameters until the pass parameter list is exhausted. An error is generated if the subprogram tries to use an optional parameter which did not have a value passed to it. The function NPAR can be used to determine the number of parameters supplied by the CALL statement invoking the subprogram.

Parameters in the formal parameter list may not be duplicated in COM statements. A subprogram may not contain any SUB statements, or DEF FN statements. Subprograms can be called recursively and may contain local variables. A unique labeled COM must be used if the local variables are to preserve their values between invocations of the subprogram

SUBEXIT may be used to leave the subprogram at some point other than the SUBEND. Multiple SUBEXITs are allowed, and SUBEXIT may appear in an IF...THEN statement. SUBEND is prohibited in IF...THEN statements, and may only occur once in a subprogram.

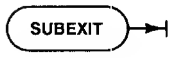
SUBEND

See the SUB statement.

SUBEXIT

Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement may be used to return from a SUB subprogram at some point other than the SUBEND statement. It allows multiple exits from a subprogram.



TAB

See the PRINT and DISP statements.

TABXY

See the PRINT statement.

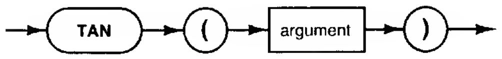
TALK

See the SEND statement.

TAN

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the tangent of the angle represented by the argument. Error 31 occurs when trying to compute the TAN of an odd multiple of 90 degrees.



Item	Description/Default	Range Restrictions
argument	numeric expression in current units of angle	absolute value less than: 8.541 563 906 E+9 deg. or 1.490 784 13 E+8 rad.

Example Statements

```
Tangent=TAN(Angle)
PRINT "Tangent of";Z;"=";TAN(Z)
```

TIMEDATE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns the current value of the real-time clock.

Example Statements

```
Elapsed=TIMEDATE-T0
DISP TIMEDATE MOD 86400
```

Semantics

The value returned by TIMEDATE represents the sum of the last time setting and the number of seconds that have elapsed since that setting was made. The clock value set at power-on is 2.086 629 12 E+11, which represents midnight March 1, 1900. The time value accumulates from that setting unless it is changed by SET TIME or SET TIMEDATE.

The resolution of the TIMEDATE function is .01 seconds. If the clock is properly set, TIMEDATE MOD 86400 gives the number of seconds since midnight.

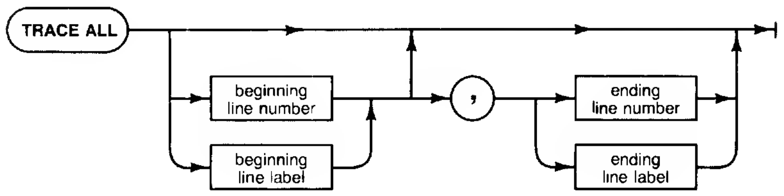
TO

See the ASSIGN and FOR...NEXT statements.

TRACE ALL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement allows tracing program flow and variable assignments during program execution.



Item	Description/Default	Range Restrictions
beginning line number	integer constant identifying a program line; Default = first program line	1 thru 32 766
beginning line label	name of a program line	any valid name
ending line number	integer constant identifying a program line; Default = last program line	1 thru 32 766
ending line label	name of a program line	any valid name

Example Statements

```
TRACE ALL Sort
TRACE ALL 1500,2450
```

Semantics

The entire program, or any part delimited by beginning and (if needed) ending line numbers or labels, may be traced.

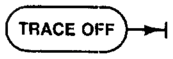
The ending line is not included in the trace output. The trace output stops immediately before the ending line is executed. When the program is traced, execution of the lines within the tracing range causes the line number and any variable which receives a new value to be output to the DISP line of the CRT. Any type of variable (string, numeric or array) can be displayed. For simple string and numeric variables, the name and the new value are displayed. For arrays, a message is displayed stating that the array has a new value rather than outputting the entire array contents.

TRACE ALL output can also be printed on the PRINTALL printer, if PRINTALL is ON. TRACE ALL is disabled by TRACE OFF. The line numbers specified for TRACE ALL are not affected by REN.

TRACE OFF

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

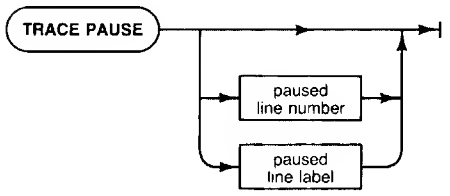
This statement turns off all tracing activity.



TRACE PAUSE

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement causes program execution to pause before executing the specified line, and displays the next line to be executed on the CRT.



Item	Description/Default	Range Restrictions
paused line number	integer constant identifying a program line; Default = next program line	1 thru 32 766
paused line label	name of a program line	any valid name

Example Statements

```
TRACE PAUSE
TRACE PAUSE Loop_end
```

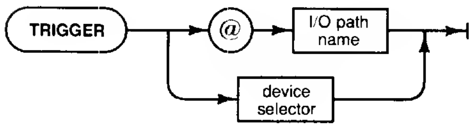
Semantics

Not specifying a line for TRACE PAUSE results in the pause occurring before the next line is executed. Only one TRACE PAUSE can be active at a time. TRACE PAUSE is cancelled by TRACE OFF.

TRIGGER

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement sends a trigger message to a selected device, or all devices addressed to listen, on the HP-IB.



Item	Description/Default	Range Restrictions
I/O path name	name assigned to a device or devices	any valid name (see ASSIGN)
device selector	numeric expression, rounded to an integer	(see Glossary)

Example Statements

```
TRIGGER 712
TRIGGER @Hpib
```

Semantics

The computer must be the active controller to execute this statement.

If only the interface select code is specified, all devices on that interface which are addressed to listen are triggered. If a primary address is given, the bus is reconfigured and only the addressed device is triggered.

Summary of Bus Actions

	System Controller		Not System Controller	
	Interface Select Code Only	Primary Addressing Specified	Interface Select Code Only	Primary Addressing Specified
Active Controller	ATN GET	ATN MTA UNL LAG GET	ATN GET	ATN MTA UNL LAG GET
Not Active Controller	Error			

UNL

See the SEND statement.

UNT

See the SEND statement.

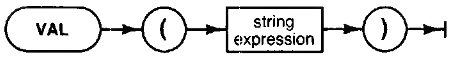
USING

See the PRINT, OUTPUT, DISP, LABEL or ENTER statement.

VAL

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function converts a string expression into a numeric value.



Item	Description/Default	Range Restrictions
argument	string expression	numerals, decimal point, sign and exponent notation

Example Statements

```
Day=VAL(Date$)
IF VAL(Response$)<0 THEN Negative
```

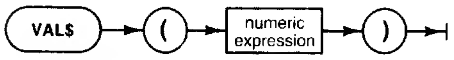
Semantics

The first non-blank character in the string must be a digit, a plus or minus sign, or a decimal point. The remaining characters may be digits, a decimal point, or an E, and must form a valid numeric constant. If an E is present, characters to the left of it must form a valid mantissa, and characters to the right must form a valid exponent. The string expression is evaluated when a non-numeric character is encountered or the characters are exhausted.

VAL$

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This function returns a string representation of the value of the argument. The returned string is in the default print format, except that the first character is not a blank for positive numbers. No trailing blanks are generated.



Item	Description/Default	Range Restrictions
argument	numeric expression	—

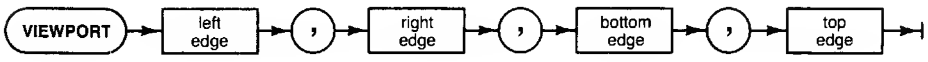
Example Statements

```
PRINT Esc$;VAL$(Cursor-1)
Special$=Text$&VAL$(Number)
```

VIEWPORT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement defines an area onto which WINDOW and SHOW statements are mapped. It also sets the soft clip limits to the boundaries it defines.



Item	Description/Default	Range Restrictions
left edge	numeric expression	—
right edge	numeric expression	—
bottom edge	numeric expression	—
top edge	numeric expression	—

Example Statements

```
VIEWPORT 0,35,50,80
VIEWPORT Left,Right,Bottom,Top
```

Semantics

The parameters for VIEWPORT are in Graphic Display Units (GDUs). Graphic Display Units are 1/100 of the shorter axis of a plotting device. The units are isotropic (the same length in X and Y).

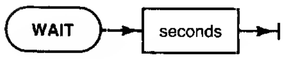
For the plotter specifier "INTERNAL" (the CRT), the shorter axis is Y. The longer axis is X, which is 133.444 816 054 GDUs long. For the plotter specifier "HPGL" (which deals with devices other than the CRT), the RATIO function may be used to determine the ratio of the length of the X axis to the length of the Y axis. If the ratio is greater than one, the Y axis is 100 GDUs long, and the length of the X axis is 100 × RATIO. If the ratio is less than one, then the length of the X axis is 100 GDUs and the length of the Y axis is 100 × RATIO. (RATIO also works with INTERNAL.)

A value of less than zero for the left edge or bottom is treated as zero. A value greater than the hard clip limit is treated as the hard clip limit for the right edge and the top. The left edge must be less than the right edge, and the bottom must be less than the top, or error 704 results.

WAIT

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement will cause the computer to wait approximately the number of seconds specified before executing the next statement. Numbers less than 0.001 do not generate a WAIT interval.



Item	Description/Default	Range Restrictions
seconds	numeric expression, rounded to the nearest thousandth	less than 2 147 483.648

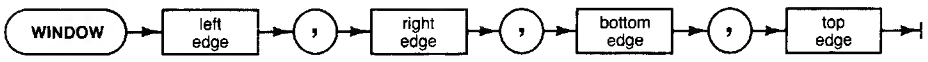
Example Statements

```
WAIT 3
WAIT Old_time/2
```

WINDOW

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement is used to define the current-unit-of-measure for graphics operations.



Item	Description/Default	Range Restrictions
left edge	numeric expression	—
right edge	numeric expression	—
bottom edge	numeric expression	—
top edge	numeric expression	—

Example Statements

```
WINDOW -5,5,0,100
WINDOW Left,Right,Bottom,Top
```

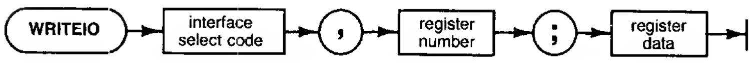
Semantics

WINDOW defines the values represented at the hard clip boundaries, or the boundaries defined by the VIEWPORT statement. WINDOW may be used to create non-isotropic (not equal in X and Y) units. The direction of an axis may be reversed by specifying the left edge greater than the right edge, or the bottom edge greater than the top edge. (Also see SHOW.)

WRITEIO

Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement writes an integer representation of the register-data to the specified hardware register on the specified interface. The actual action resulting from this operation depends on the interface and register selected.



Item	Description/Default	Range Restrictions	Recommended Range
interface select code	numeric expression, rounded to an integer	1 thru 31	—
register number	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	interface dependent
register data	numeric expression, rounded to an integer	-2^{31} thru $+2^{31}-1$	−32 768 thru +32 767

Note

Unexpected and possibly undesirable results may occur with select codes outside the given range.

Example Statements

```
WRITEIO 12,0;Set_pctl
WRITEIO Hpib,23;12
```

Glossary

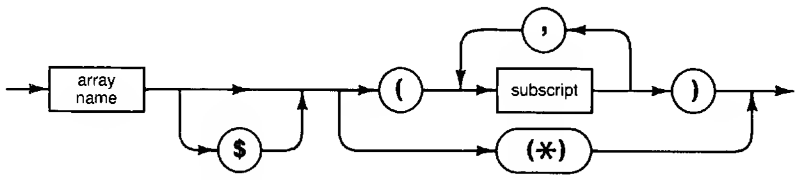
angle mode The current units used for expressing angles. Either degrees or radians may be specified, using the DEG or RAD statements, respectively. The default at power-on and SCRATCH A is radians.

A subprogram "inherits" the angle mode of the calling context. If the angle mode is changed in a subprogram, the mode of the calling context is restored when execution returns to the calling context.

array A structured data type that can be of type REAL, INTEGER, or string. Arrays are created with the DIM, REAL, INTEGER, ALLOCATE, or COM statements. Arrays have 1 to 6 dimensions; each dimension is allowed 32 767 elements. The lower and upper bounds for each dimension must fall in the range −32 767 (−32 768 for ALLOCATE) thru +32 767, and the lower bound must not exceed the upper bound. The default lower bound is the OPTION BASE value; the OPTION BASE statement can be used to specify 0 or 1 as the default lower bound. The default OPTION BASE at power-on or SCRATCH A is zero.

Each element in a string array is a string whose maximum length is specified in the declaring statement. The declared length of a string must be in the range 1 thru 32 767.

To specify an entire array, the characters (*) are placed after the array name. To specify a single element of an array, subscripts are placed in parentheses after the array name. Each subscript must not be less than the lower bound or greater than the upper bound of the corresponding dimension.



If an array is not explicitly dimensioned, it is implicitly given the number of dimensions used in its first occurrence, with an upper bound of 10. Undeclared strings have a default length of 18.

ASCII This is the acronym for "American Standard Code for Information Interchange". It is a commonly used code for representing letters, numerals, punctuation, special characters, and control characters. A table of the characters in the ASCII set and their code values can be found in the back of this manual.

bit This term comes from the words "binary digit". A bit is a single digit in base 2 that must be either a 1 or a 0.

byte A group of eight bits processed as a unit.

command A statement that can be typed on the input line and executed (see "statement").

context An instance of an environment. A context consists of a specific instance of all data types which may be accessed by a program at a specific point in its execution.

device selector A numeric expression used to specify the source or destination of an I/O operation. A device selector can be either an interface select code or a combination of an interface select code and a primary address. To construct a device selector with a primary address, multiply the interface select code by 100 and add the primary address.

Secondary addresses may be appended after a primary address by multiplying the device selector by 100 and adding the address. This may be repeated up to 6 times, adding a new secondary address each time. A device selector, once rounded, can contain a maximum of 15 digits.

When a device selector contains an odd number of digits, the leftmost digit is the interface select code. For an even number of digits, the leftmost two digits are the interface select code. For example, 70502 selects interface 7, primary address 05, and secondary address 02. Device selector 1516 selects interface 15 and primary address 16.

dyadic operator An operator that performs its operation on two expressions. It is placed between the two expressions. The following dyadic operators are available:

Operator	Operation
+	REAL or INTEGER addition
-	REAL or INTEGER subtraction
*	REAL or INTEGER multiplication
/	REAL division
^	Exponentiation
&	String concatenation
DIV	Gives the integer quotient of a division
MOD	Gives the integer remainder (modulus) of a division
=	Comparison for equality
<>	Comparison for inequality
<	Comparison for less than
>	Comparison for greater than
<=	Comparison for less than or equal to
>=	Comparison for greater than or equal to
AND	Logical AND (Boolean $\wedge$)
OR	Logical inclusive OR (Boolean $\vee$)
EXOR	Logical exclusive OR (Boolean $\veebar$)

file name A file name consists of one to ten characters. 9826 file names can contain uppercase letters, lowercase letters, numerals, the underbar (_), and CHR$(161) thru CHR$(254). LIF-compatible file names can contain only uppercase letters and numerals. The first character in a LIF-compatible file name must be a letter.

function A procedural call that returns a value. The call can be to a user-defined-function subprogram (such as FNInvert) or a machine-resident function (such as COS or EXP). The value returned by the function is used in place of the function call when evaluating the expression containing the function call.

graphic display unit This is 1/100 of the shortest axis on the plotting device. Graphic display units are the same size on both the X and Y axes. Abbreviated "GDU".

hard clip limits These are the physical limits of the plotting device.

hierarchy When a numeric or string expression contains more than one operation, the order of operations is determined by a precedence system. Operations with the highest precedence are performed first. Multiple operations with the same precedence are performed in order, left to right. The following tables show the hierarchy for numeric and string operations.

Math Hierarchy

Precedence	Operator
Highest	Parentheses; they may be used to force any order of operations.
	Functions, both user-defined and machine-resident
	Exponentiation: ^
	Multiplication and division: * / MOD DIV
	Addition, subtraction, monadic plus and minus: + -
	Relational operators: = <> < > <= >=
	NOT
	AND
Lowest	OR EXOR

String Hierarchy

Precedence	Operator
Highest	Parentheses
	Functions, both user-defined and machine-resident
Lowest	Concatenation: &

I/O path A combination of firmware and hardware that can be used during the transfer of data to and from a BASIC program. Associated with an I/O path is a unique data type that describes the I/O path. This association table uses about 200 bytes and is referenced by an I/O path name. For further details, see the ASSIGN statement.

INTEGER A numeric data type stored internally in two bytes. Two's-complement representation is used, giving a range of −32 768 thru +32 767.

If a numeric variable is not explicitly declared as an INTEGER, it is a REAL.

integer A number with no fractional part; a whole number.

interface select code A numeric expression that selects an interface for an I/O operation. Interface select codes 1 thru 7 are reserved for internal interfaces. Interface select codes 8 thru 31 are used for external interfaces. The internal HP-IB interface with select code 7 can be specified in statements that are restricted to external interfaces. (Also see "device selector".)

keyword A group of uppercase ASCII letters that has a predefined meaning to the computer. Keywords may be typed using all lowercase or all uppercase letters.

LIF This is the acronym for "Logical Interchange Format". This HP standard defines the format of mass storage files and directories. It allows the interchange of data between different machines. 9826 files of type ASCII are LIF compatible.

literal This is a string constant. When quote marks are used to delimit a literal, those quote marks are not part of the literal. To include a quote mark in a literal, type two consecutive quote marks (except in response to a LINPUT statement). The drawings showing literal forms of specifiers (such as file specifiers) show the quote marks required to delimit the literal.

logical pen See "pen".

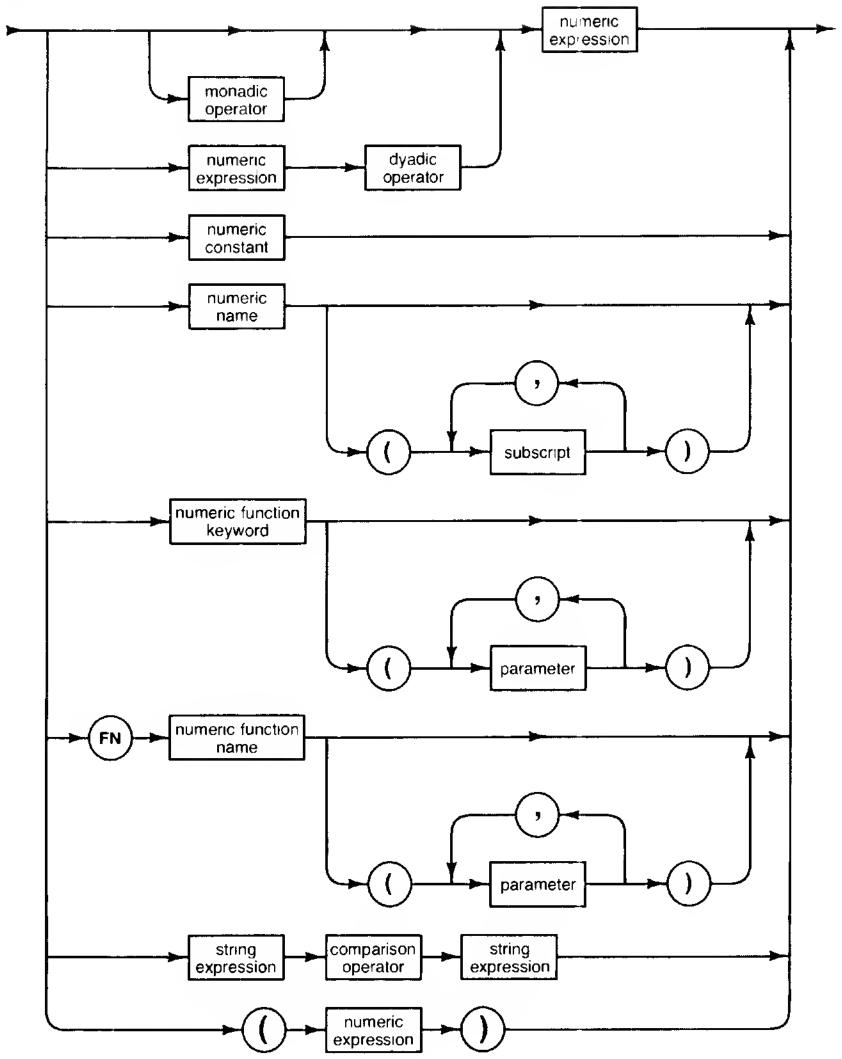
monadic operator An operator that performs its operation on one expression. It is placed in front of the expression. The following monadic operators are available:

Operator	Operation
-	Reverses the sign of an expression
+	Identity operator
NOT	Logical complement (Boolean over-bar)

msus This is the acronym for "mass storage unit specifier". It is a string expression that specifies a device to be used for mass storage operations.

name A name consists of one to fifteen characters. The first character must be an uppercase ASCII letter or one of the characters from CHR$(161) thru CHR$(254). The remaining characters, if any, can be lowercase ASCII letters, numerals, the underbar (_), or CHR$(161) thru CHR$(254). Names may be typed using any combination of uppercase and lowercase letters, unless the name uses the same letters as a keyword. Conflicts with keywords are resolved by mixing the letter case in the name. (Also see "file name".)

numeric expression



Item	Description
monadic operator	An operator that performs its operation on the expression immediately to its right: `+ - NOT`
dyadic operator	An operator that performs its operation on the two expressions it is between: `^ * / MOD DIV + - = <> < > <= >= AND OR EXOR`
numeric constant	A numeric quantity whose value is expressed using numerals, decimal point, and exponent notation
numeric name	The name of a numeric variable or the name of a numeric array from which an element is extracted using subscripts
subscript	A numeric expression used to select an element of an array (see "array")
numeric function keyword	A keyword that invokes a machine-resident function that returns a numeric value
numeric function name	The name of a user-defined function that returns a numeric value
parameter	A numeric expression, string expression, or I/O path name that is passed to a function
comparison operator	An operator which returns a 1 (true) or a 0 (false) based on the the result of a relational test of the operands it separates: `> < <= >= = <>`

pen All graphical objects are "drawn" using mathematical representations in the computer's memory. This is done with the "logical pen". The logical pen creates four classes of objects: lines, labels, axes, and label locations (label locations are actually the position of an object, rather than an object).

Before these objects can be viewed, they are acted upon by various transformation matrixes, such as scaling and pivoting. No single transformation affects all the objects, and no object is effected by all the transformations.

The output of the transformations is used to control the "physical pen". The physical pen creates the image that you actually see on the plotter or CRT. Since the graphics statements used to create objects act directly upon the logical pen, and you can see only the output of the physical pen, the location of the logical pen may not always be readily discernable from what you see.

The following table shows which transformations act upon which objects.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

primary address A numeric expession in the range of 0 thru 31 that specifies an individual device on an interface which is capable of servicing more than one device. The HP-IB interface can service multiple devices. (Also see "device selector".)

program line A statement that is preceded by a line number (and an optional line label) and stored with the ENTER key into a program (see "statement").

protect code This is a non-listable, two-character code kept with a file description in the directory of a mass storage media. It guards against accidental changes to an individual file. When protect codes are specified, they may contain any number of characters. When the specified protect code contains more than two characters, the first two are used as the actual protect code. When the specified protect code contains only one character, a blank is assumed as the second character in the actual protect code. A protect code that is all blanks, the null string, or begins with at least two blanks is interpreted the same as no protect code. The character > is not allowed in a protect code.

REAL A numeric data type that is stored internally in eight bytes using sign-and-magnitude representation. One bit is used for the number's sign, 11 bits for a biased exponent (bias = 1023), and 52 bits for a mantissa. There is an implied "1." preceding the mantissa (this can be thought of as the 53rd bit). The range of REAL numbers is approximately:

−1.797 693 134 862 315 E+308 thru −2.225 073 858 507 202 E−308, 0, and +2.225 073 858 507 202 E−308 thru +1.797 693 134 862 315 E+308.

If a variable is not explicitly declared as an INTEGER, it is a REAL.

record The records referred to in this manual are defined records. Defined records are the smallest unit of storage directly accessible on the mass storage media. The length of a record is determined when a BDAT file is created by a CREATE BDAT statement. All records in a file are the same size.

There is another type of record called a "physical record" which is the unit of storage handled by the mass storage device and the operating system. Physical records contain 256 bytes and are not accessible to the user via standard BASIC statements.

recursive See "recursive".

secondary address A device-dependent command sent on HP-IB. It can be interpreted as a secondary address for the extended talker/listener functions or as part of a command sequence. (Also see ''device selector''.)

soft clip limits These are plotter clipping limits that are defined by the programmer. Lines drawn on a plotting device are drawn only inside the clipping limits.

statement A keyword combined with any additional items that are allowed or required with that keyword. If a statement is placed after a line number and stored, it becomes a program line. If a statement is typed without a line number and executed, it is called a command.

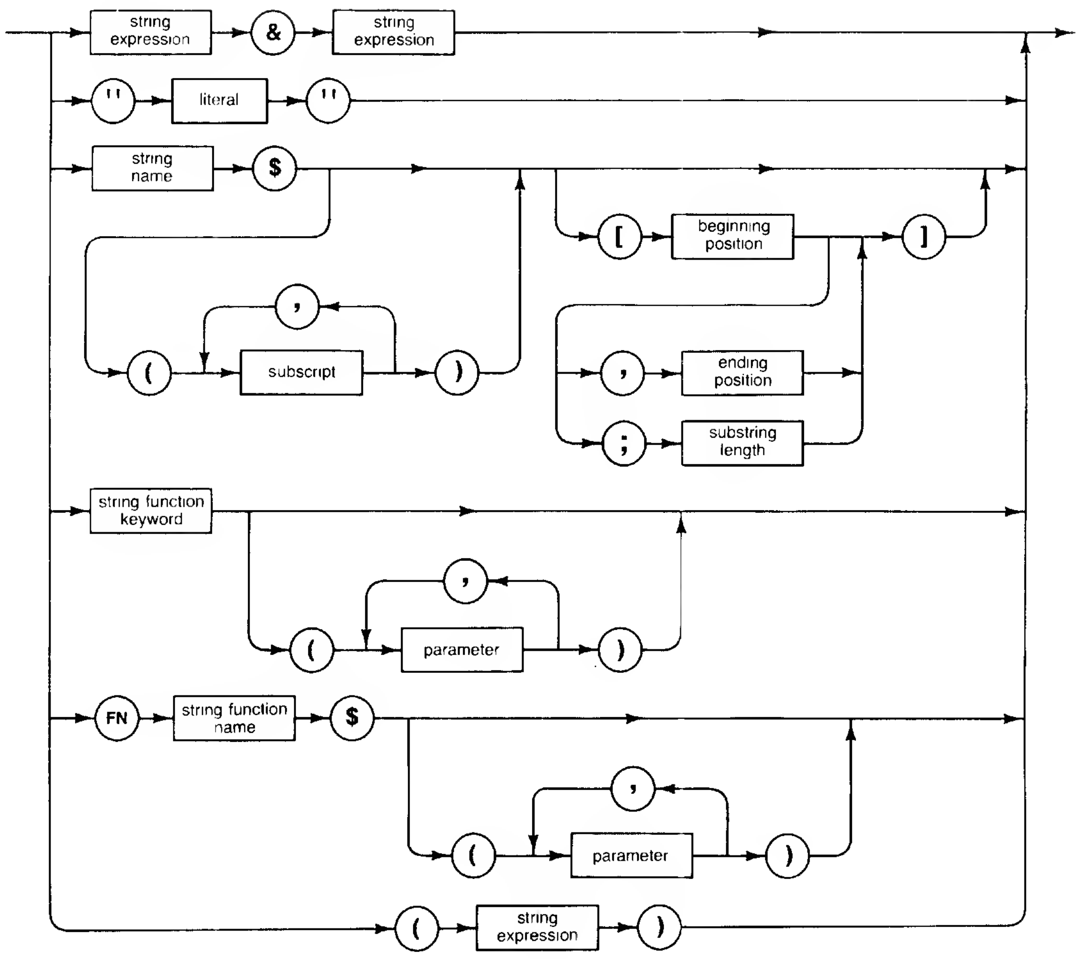
string A data type comprised of a contiguous series of characters. Each character in the string is stored in one byte using an extended ASCII character set. The first character in a string is in position 1. The maximum length of a string is 32 767 characters. The current length of a string can never exceed the dimensioned length.

If a string is not explicitly dimensioned, it is implicitly dimensioned to 18 characters. Each element in an implicitly dimensioned string array is dimensioned to 18 characters.

When a string is empty, it has a current length of zero and is called a ''null string''. All strings are null strings when they are declared. A null string can be represented as an empty literal (for example: `A$=""`) or as one of three special cases of substring. The substrings that represent the null string are:

1. Beginning position one greater than current length
2. Ending position one less than beginning position
3. Maximum substring length of zero

string expression



Item	Description
literal	A string constant composed of any characters available on the keyboard, including those generated with the ANY CHAR key.
string name	The name of a string variable or the name of a string array from which a string is extracted using subscripts
subscript	A numeric expression used to select an element of an array (see "array")
beginning position	A numeric expression specifying the position of the first character in a substring (see "substring")
ending position	A numeric expression specifying the position of the last character in a substring (see "substring")
substring length	A numeric expression specifying the maximum number of characters to be included in a substring (see "substring")
string function keyword	A keyword that invokes a machine-resident function that returns a string value. String function keywords always end with a dollar sign.
string function name	The name of a user-defined function that returns a string value
parameter	A numeric expression, string expression, or I/O path name that is is passed to a function

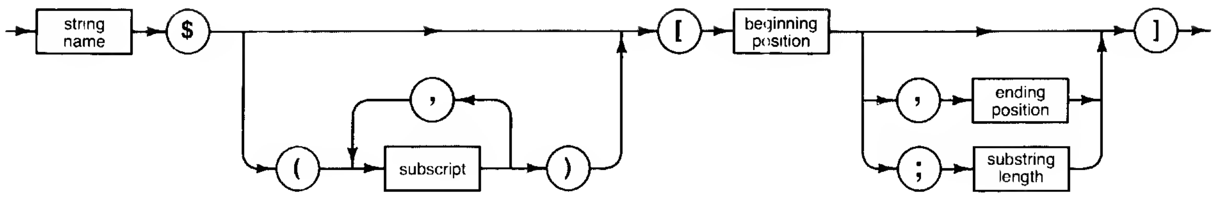
subprogram Can be either a SUB subprogram or a user-defined-function subprogram (DEF FN). The first line in a SUB subprogram is a SUB statement. The last line in a SUB subprogram (except for comments) is a SUBEND statement. The first line in a function subprogram is a DEF FN statement. The last line in a function (except for comments) is an FNEND statement. Subprograms must follow the END statement of the main program.

SUB subprograms are invoked by CALL. Function subprograms are invoked by an FN function occurring in an expression. A function subprogram returns a value that replaces the occurrence of the FN function when the expression is evaluated. Either type of subprogram may alter the values of parameters passed by reference or variables in COM. It is recommended that you do not let function subprograms alter values in that way.

Invoking a subprogram establishes a new context. The new context remains in existence until the subprogram is properly exited or program execution is stopped. Subprograms can be recursive.

subroutine A program segment accessed by a GOSUB statement and ended with a RETURN statement.

substring



A substring is a contiguous series of characters that comprises all or part of a string. Substrings may be accessed by specifying a beginning position, or a beginning position and an ending position, or a beginning position and a maximum substring length.

The beginning position must be at least one and no greater than the current length plus one. When only the beginning position is specified, the substring includes all characters from that position to the current end of the string.

The ending position must be no less than the beginning position minus one and no greater than the dimensioned length of the string. When both beginning and ending positions are specified, the substring includes all characters from the beginning position to the ending position or current end of the string, whichever is less.

The maximum substring length must be at least zero and no greater than one plus the dimensioned length of the string minus the beginning position. When a beginning position and substring length are specified, the substring starts at the beginning position and includes the number of characters specified by the substring length. If there are not enough characters available, the substring includes only the characters from the beginning position to the current end of the string.

Interface Registers

I/O Path Status and Control Registers

Status Register 0	0 = Invalid I/O path name 1 = I/O path name assigned to a device 2 = I/O path name assigned to a data file

If Assigned to a Device:

Status Register 1	Interface select code
Status Register 2	Number of devices
Status Register 3	1st device selector

If assigned to more than one device, the other device selectors are available starting in Status Register 4.

If Assigned to an ASCII file:

Status Register 1	File type = 3
Status Register 2	Device selector of mass storage device
Status Register 3	Number of physical records
Status Register 4	Bytes per record = 256
Status Register 5	Current physical record
Status Register 6	Current byte within physical record

If Assigned to a BDAT file:

Status Register 1	File type = 2
Status Register 2	Device selector of mass storage device
Status Register 3	Number of defined records
Status Register 4	Defined record length (bytes)
Status Register 5	Current record
Control Register 5	Set current record
Status Register 6	Current byte within record
Control Register 6	Set current byte within record
Status Register 7	EOF record
Control Register 7	Set EOF record
Status Register 8	Byte within EOF record
Control Register 8	Set byte within EOF record

CRT Status and Control Registers

Status Register 0	Current PRINT position (X)
Control Register 0	Set PRINT position (X)
Status Register 1	Current PRINT position (Y)
Control Register 1	Set PRINT position (Y)
Status Register 2	Insert character mode
Control Register 2	Set insert character mode if non-0
Status Register 3	Number of lines in offscreen memory above top of CRT.
Control Register 3	Undefined
Status Register 4	Display functions mode
Control Register 4	Set display functions mode if non-0
Status Register 5	Undefined
Control Register 5	Undefined
Status Register 6	ALPHA ON flag
Control Register 6	Undefined
Status Register 7	GRAPHICS ON flag
Control Register 7	Undefined
Status Register 8	Display line position (X)
Control Register 8	Set display line position (X)
Status Register 9	Screenwidth (number of characters)
Control Register 9	Undefined
Status Register 10	Cursor enable flag
Control Register 10	Cursor enable; 0 = cursor not visible non-0 = cursor visible
Status Register 11	CRT character mapping flag
Control Register 11	Disable CRT character mapping if non-0

Keyboard Status and Control Registers

Status Register 0	CAPS LOCK flag
Control Register 0	Set CAPS LOCK if non-0
Status Register 1	PRINTALL flag
Control Register 1	Set PRINTALL if non-0
Status Register 2	Undefined
Control Register 2	Undefined
Status Register 3	Undefined
Control Register 3	Set auto-repeat interval.If 1 thru 255, repeat rate in milliseconds is 10 times this value. 256 = turn off auto-repeat. (Default at power-on or SCRATCH A is 80 ms.)
Status Register 4	Undefined
Control Register 4	Set delay before auto-repeat. If 1 thru 256, delay in milliseconds is 10 times this value. (Default at power-on or SCRATCH A is 700 ms.)
Status Register 5	Undefined
Control Register 5	Undefined
Status Register 6	Undefined
Control Register 6	Undefined

Status Register 7 **Interrupt Status**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	INITIALIZE Timeout Interrupt Disabled	Reserved For Future Use	Reserved For Future Use	RESET Key Interrupt Disabled	Keyboard and Knob Interrupt Disabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 7 (Set bit to disable) **Interrupt Disable Mask**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			INITIALIZE Timeout	Reserved For Future Use	Reserved For Future Use	RESET Key	Keyboard and Knob
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 8	Keyboard language jumper 0 = US ASCII 1 = French 2 = German 3 = Swedish/Finnish 4 = Spanish 5 = Katakana
Control Register 8	Undefined
Status Register 9	Keyboard configuration jumper (0 thru 8)
Control Register 9	Undefined

Status Register 10 **State at Last Knob Interrupt**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	CTRL Key Pressed	SHIFT Key Pressed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 10	Undefined
Status Register 11	Reserved for future use
Control Register 11	Undefined
Status Register 12	"Pseudo-EOI for CTRL-E" flag
Control Register 12	Enable pseudo-EOI for CTRL-E if non-0
Status Register 13	Katakana flag
Control Register 13	Set Katakana if non-0

HP-IB Status and Control Registers

Status Register 0 Card identification = 1

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrrupts Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 **Serial Poll Response Byte**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Dependent Status	SRQ 1 = I did it 0 = I didn't	Device Dependent Status					
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 2

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Reserved For Future Use	Handshake In Progress	Interrupts Enabled	Reserved For Future Use
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 **Parallel Poll Response Byte**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8 1 = True	DIO7 1 = True	DIO6 1 = True	DIO5 1 = True	DIO4 1 = True	DIO3 1 = True	DIO2 1 = True	DIO1 1 = True
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 **Controller Status and Address**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
System Controller	Active Controller	0	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 3 **Set My Addrress**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used			Primary Address				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 4 **Interrupt Status**

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 4 Writing anything to this register releases NDAC holdoff. If non-zero, accept last secondary address as valid. If zero, don't accept last secondary address (stay in LPAS or TPAS state).

Status Register 5 **Interrupt Enable Mask**

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 1Ø	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 6 **Interface Status**

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 1Ø	Bit 9	Bit 8
REM	LLO	ATN True	LPAS	TPAS	LADS	TADS	*
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
System Controller	Active Controller	Ø	Primary Address of Interface				
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Least-significant bit of last address recognized

Status Register 7

Bus Control and Data Lines

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
ATN True	DAV True	NDAC* True	NRFD* True	EOI True	SRQ** True	IFC True	REN True
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DIO8	DIO7	DIO6	DIO5	DIO4	DIO3	DIO2	DIO1
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* Only if addressed to TALK, else not valid.
** Only if Active Controller, else not valid.

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Active Controller	Parallel Poll Configuration Change	My Talk Address Received	My Listen Address Received	EOI Received	SPAS	Remote/ Local Change	Talker/ Listener Address Change
Value = −32 768	Value = 16 384	Value = 8 192	Value = 4 096	Value = 2 048	Value = 1 024	Value = 512	Value = 256

Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Trigger Received	Handshake Error	Unrecognized Universal Command	Secondary Command While Addressed	Clear Received	Unrecognized Addressed Command	SRQ Received	IFC Received
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

RS-232 Status and Control Registers

Status Register 0 — **Card Identification**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1 = Future Use Jumper Installed	0	0	0	0	0	1	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 0 Reset interface if non-zero

Status Register 1 — **Interrupt Status**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupts Enabled	Interrupt Requested	Hardware Interrupt Level Switches		0	0	0	0
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Send break if non-zero

Status Register 2

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Progress	Interrupts Enabled	Reserved For Future Use
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2	Undefined
Status Register 3	Baud Rate
Control Register 3	Set Baud Rate

Status Register 4 — **Character Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Reserved for Future Use		ØØ = Odd Parity Ø1 = Even Parity 1Ø = Parity Bit "1" 11 = Parity Bit "Ø"		Parity Enabled	Ø = One Stop Bit 1 = Two Stop Bits*	Character Length (add this value to 5)	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*1.5 stop bit if character length is 5.

Control Register 4 — **Character Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Not Used		ØØ = Odd parity Ø1 = Even Parity 1Ø = Parity Bit "1" 11 = Parity Bit "Ø"		1 = Enable parity	Ø = One Stop Bit 1 = Two Stop Bits*	Character Length (add this value to 5)	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

*1.5 stop bits if character length is 5.

Status Register 5 — **Modem Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Ø	Ø	Ø	Loop Back Mode	Secondary Request To Send	Data Rate Select	Request To Send	Data Terminal Ready
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 5 — **Modem Control**

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit Ø
Not Used			1 = Set Loopback Mode	1 = Set Secondary Request To Send	1 = Set Data Rate Select	RTS* 1 = Set Ø = Handshake	DTR** 1 = Set Ø = Handshake
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

* 0 = Set only during an OUTPUT statement.

** 0 = Set only during an OUTPUT or ENTER statement.

Status Register 6 Data In (8 bits)

Control Register 6 Data Out (8 bits)

Status Register 7 **Optional Circuits**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Optional Driver 3	Optional Driver 4	Optional Receiver 2	Optional Receiver 3
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 7 **Optional Circuits**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Optional Driver 3	Optional Driver 4	Not Used	
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 8 **Interrupt Enable Mask**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 9 **Interrupt Cause**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	11 = Receiver Line Status 10 = Receiver Buffer Full 01 = Transmitter Holding Register Empty 00 = Modem Status Change		0 = UART Requesting Interrupt
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 10 — UART Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Transmit Shift Register Empty	Transmit Holding Register Empty	Break Received	Framing Error	Parity Error	Overrun Error	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 11 — Modem Status

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Carrier Detect	Ring Indicator	Data Set Ready	Clear To Send	Change In Carrier Detect	Ring Indicator Changed To False	Change In Data Set Ready	Change In Clear To Send
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used				Modem Status Change	Receiver Line Status	Transmitter Holding Register Empty	Receiver Buffer Full
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

GPIO Status and Control Registers

Status Register 0 Card identification = 3

Control Register 0 Reset interface if non-zero

Status Register 1 **Interrupt and DMA Status**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Interrupt Enabled	Interrupt Requested	Hardware Interrupt Level Switches		DMA Burst Mode	DMA Word Mode	DMA Channel 1 Enabled	DMA Channel 0 Enabled
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 1 Set PCTL if non-zero

Status Register 2

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	Handshake In Process	Interrupts Enabled	Reserved For Future Use
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Control Register 2 **Peripheral Control**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					PSTS Error 1 = Report 0 = Ignore	Set CTL1 1 = Low 0 = High	Set CTL0 1 = Low 0 = High
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Status Register 3 Data In (16 bits)

Control Register 3 Data Out (16 bits)

Status Register 4 1 = Ready; 0 = Busy

Status Register 5 **Peripheral Status**

Most Significant Bit / Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	PSTS Line True	EIR Line Low	STI1 Low	STI0 Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Interrupt Enable Register (ENABLE INTR)

Most Significant Bit / Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used						PFLG Line Ready	EIR Line Low
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Useful Tables

Interface Select Codes

Internal Select Codes

1	Display (alpha)
2	Keyboard
3	Display (graphics)
4	(not used)
5	(not used)
6	(not used)
7	HP-IB interface (built-in)

Factory Presets for External Interfaces

8	HP-IB
9	RS-232
10	(not used)
11	BCD
12	GPIO

US ASCII Character Codes

ASCII	EQUIVALENT FORMS				HP-IB
Char.	Blnary	Oct	Hax	Dac	
NULL	00000000	000	00	0	
SOH	00000001	001	01	1	GTL
STX	00000010	002	02	2	
ETX	00000011	003	03	3	
EOT	00000100	004	04	4	SDC
ENQ	00000101	005	05	5	PPC
ACK	00000110	006	06	6	
BELL	00000111	007	07	7	
BS	00001000	010	06	6	GET
HT	00001001	011	09	9	TCT
LF	00001010	012	0A	10	
VT	00001011	013	0B	11	
FF	00001100	014	0C	12	
CR	00001101	015	0D	13	
SO	00001110	016	0E	14	
SI	00001111	017	0F	15	
DLE	00010000	020	10	16	
DC1	00010001	021	11	17	LLO
DC2	00010010	022	12	16	
DC3	00010011	023	13	19	
DC4	00010100	024	14	20	DCL
NAK	00010101	025	15	21	PPU
SYNC	00010110	026	16	22	
ETB	00010111	027	17	23	
CAN	00011000	030	16	24	SPE
EM	00011001	031	19	25	SPD
SUB	00011010	032	1A	26	
ESC	00011011	033	1B	27	
FS	00011100	034	1C	26	
GS	00011101	035	1D	29	
RS	00011110	036	1E	30	
US	00011111	037	1F	31	

ASCII	EQUIVALENT FORMS				HP-IB
Char.	Blnary	Oct	Hax	Dac	
space	00100000	040	20	32	LA0
!	00100001	041	21	33	LA1
"	00100010	042	22	34	LA2
#	00100011	043	23	35	LA3
$	00100100	044	24	36	LA4
%	00100101	045	25	37	LA5
&	00100110	046	26	36	LA6
'	00100111	047	27	39	LA7
(	00101000	050	26	40	LA6
)	00101001	051	29	41	LA9
*	00101010	052	2A	42	LA10
+	00101011	053	2B	43	LA11
,	00101100	054	2C	44	LA12
–	00101101	055	2D	45	LA13
.	00101110	056	2E	46	LA14
/	00101111	057	2F	47	LA15
0	00110000	060	30	46	LA16
1	00110001	061	31	49	LA17
2	00110010	062	32	50	LA16
3	00110011	063	33	51	LA19
4	00110100	064	34	52	LA20
5	00110101	065	35	53	LA21
6	00110110	066	36	54	LA22
7	00110111	067	37	55	LA23
6	00111000	070	36	56	LA24
9	00111001	071	39	57	LA25
:	00111010	072	3A	56	LA26
;	00111011	073	3B	59	LA27
<	00111100	074	3C	60	LA26
=	00111101	075	3D	61	LA29
>	00111110	076	3E	62	LA30
?	00111111	077	3F	63	UNL

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
@	01000000	100	40	64	TA0
A	01000001	101	41	65	TA1
B	01000010	102	42	66	TA2
C	01000011	103	43	67	TA3
D	01000100	104	44	66	TA4
E	01000101	105	45	69	TA5
F	01000110	106	46	70	TA6
G	01000111	107	47	71	TA7
H	01001000	110	46	72	TA6
I	01001001	111	49	73	TA9
J	01001010	112	4A	74	TA10
K	01001011	113	4B	75	TA11
L	01001100	114	4C	76	TA12
M	01001101	115	4D	77	TA13
N	01001110	118	4E	78	TA14
O	01001111	117	4F	79	TA15
P	01010000	120	50	80	TA16
Q	01010001	121	51	61	TA17
R	01010010	122	52	82	TA16
S	01010011	123	53	63	TA19
T	01010100	124	54	64	TA20
U	01010101	125	55	85	TA21
V	01010110	126	56	66	TA22
W	01010111	127	57	87	TA23
X	01011000	130	58	88	TA24
Y	01011001	131	59	69	TA25
Z	01011010	132	5A	90	TA26
[	01011011	133	5B	91	TA27
\	01011100	134	5C	92	TA26
]	01011101	135	5D	93	TA29
^	01011110	136	5E	94	TA30
_	01011111	137	5F	95	UNT

ASCII Char.	EQUIVALENT FORMS				HP-IB
	Binary	Oct	Hex	Dec	
`	01100000	140	60	96	SC0
a	01100001	141	61	97	SC1
b	01100010	142	62	96	SC2
c	01100011	143	63	99	SC3
d	01100100	144	64	100	SC4
e	01100101	145	85	101	SC5
f	01100110	146	66	102	SC6
g	01100111	147	67	103	SC7
h	01101000	150	66	104	SC6
i	01101001	151	69	105	SC9
j	01101010	152	6A	106	SC10
k	01101011	153	6B	107	SC11
l	01101100	154	6C	108	SC12
m	01101101	155	6D	109	SC13
n	01101110	156	6E	110	SC14
o	01101111	157	6F	111	SC15
p	01110000	160	70	112	SC16
q	01110001	161	71	113	SC17
r	01110010	162	72	114	SC18
s	01110011	163	73	115	SC19
t	01110100	164	74	116	SC20
u	01110101	165	75	117	SC21
v	01110110	166	76	118	SC22
w	01110111	167	77	119	SC23
x	01111000	170	78	120	SC24
y	01111001	171	79	121	SC25
z	01111010	172	7A	122	SC26
{	01111011	173	7B	123	SC27
|	01111100	174	7C	124	SC26
}	01111101	175	7D	125	SC29
~	01111110	176	7E	126	SC30
DEL	01111111	177	7F	127	SC31

European Display Characters

Character	Decimal Value	Character	Decimal Value	Character	Decimal Value
hp	128	hp	173	Ö	218
hp	129	hp	174	Ü	219
hp	130	₤	175	É	220
hp	131	‾	176	ï	221
hp	132	hp	177	ß	222
hp	133	hp	178	hp	223
hp	134	°	179	hp	224
hp	135	hp	180	hp	225
hp	136	Ç	181	hp	226
hp	137	Ñ	182	hp	227
hp	138	ñ	183	hp	228
hp	139	¡	184	hp	229
hp	140	¿	185	hp	230
hp	141	¤	186	hp	231
hp	142	£	187	hp	232
hp	143	hp	188	hp	233
hp	144	§	189	hp	234
hp	145	hp	190	hp	235
hp	146	hp	191	hp	236
hp	147	â	192	hp	237
hp	148	ê	193	hp	238
hp	149	ô	194	hp	239
hp	150	û	195	hp	240
hp	151	á	196	hp	241
hp	152	é	197	hp	242
hp	153	ó	198	hp	243
hp	154	ú	199	hp	244
hp	155	à	200	hp	245
hp	156	è	201	hp	246
hp	157	ò	202	hp	247
hp	158	ù	203	hp	248
hp	159	ä	204	hp	249
hp	160	ë	205	hp	250
hp	161	ö	206	hp	251
hp	162	ü	207	hp	252
hp	163	Å	208	hp	253
hp	164	î	209	hp	254
hp	165	Ø	210	🄺	255
hp	166	Æ	211		
hp	167	å	212		
´	168	í	213		
`	169	ø	214		
ˆ	170	æ	215		
¨	171	Ä	216		
˜	172	ì	217		

Katakana Display Characters

Character	Decimal Value	Character	Decimal Value	Character	Decimal Value
lF	128	ｭ	173	ﾚ	218
lF	129	ｮ	174	ﾛ	219
lF	130	ｯ	175	ﾜ	220
lF	131	ｰ	176	ﾝ	221
lF	132	ｱ	177	ﾞ	222
lF	133	ｲ	178	ﾟ	223
lF	134	ｳ	179	lF	224
lF	135	ｴ	180	lF	225
lF	136	ｵ	181	lF	226
lF	137	ｶ	182	lF	227
lF	138	ｷ	183	lF	228
lF	139	ｸ	184	lF	229
lF	140	ｹ	185	lF	230
lF	141	ｺ	186	lF	231
lF	142	ｻ	187	lF	232
lF	143	ｼ	188	lF	233
lF	144	ｽ	189	lF	234
lF	145	ｾ	190	lF	235
lF	146	ｿ	191	lF	236
lF	147	ﾀ	192	lF	237
lF	148	ﾁ	193	lF	238
lF	149	ﾂ	194	lF	239
lF	150	ﾃ	195	lF	240
lF	151	ﾄ	196	lF	241
lF	152	ﾅ	197	lF	242
lF	153	ﾆ	198	lF	243
lF	154	ﾇ	199	lF	244
lF	155	ﾈ	200	lF	245
lF	156	ﾉ	201	lF	246
lF	157	ﾊ	202	lF	247
lF	158	ﾋ	203	lF	248
lF	159	ﾌ	204	lF	249
lF	160	ﾍ	205	lF	250
｡	161	ﾎ	206	lF	251
｢	162	ﾏ	207	lF	252
｣	163	ﾐ	208	lF	253
､	164	ﾑ	209	lF	254
･	165	ﾒ	210	🄺	255
ｦ	166	ﾓ	211		
ｧ	167	ﾔ	212		
ｨ	168	ﾕ	213		
ｩ	169	ﾖ	214		
ｪ	170	ﾗ	215		
ｫ	171	ﾘ	216		
ｬ	172	ﾙ	217		

Master Reset Table

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/ STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
CRT														
CRT DISP Line	Clear	Clear	—	—	Clear	—	—	—	—	—	—	—	—	—
CRT Display Functions	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
CRT Message Line	Ready	Clear	Clear	Clear	Reset	—	—	—	—	—	—	Clear	—	—
CRT Input Line (Note 6)	Clear	Clear	Clear	—	Clear	—	—	—	—	—	—	—	—	—
CRT Printout Area	Clear	Clear	—	—	—	—	—	—	—	—	—	—	—	—
CRT Print Position (TABXY)	1,1	1,1	—	—	—	—	—	—	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	On	—	On	—	—	—	—	—
KEYBOARD														
Keyboard Recall Buffer	Clear	—	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Result Buffer	Empty	Empty	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Knob Mode	↕	↕	↕	↕	↕	—	—	—	—	—	—	—	—	—
Tabs On Input Line	None	None	—	—	—	—	—	—	—	—	—	—	—	—
Keyboard Katakana Mode	Off	Off	Off	—	Off	—	—	—	—	—	—	—	—	—
PRINTING														
Print column	1	1	—	—	1	—	—	—	—	—	—	—	—	—
PRINTALL	Off	Off	—	—	Off	—	—	—	—	—	—	—	—	—
PRINTALL IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
PRINTER IS	1	1	—	—	—	—	—	—	—	—	—	—	—	—
ENVIRONMENTS & VARIABLES														
Allocated Variables	None	None	None	None	Note 1	Note 1	None	None	None	None	—	None	None	Pre-ent
Normal Variables	None	None	None	None	—	—	None	None	None	None	—	Note 11	Note 11	Pre-ent
COM Variables	None	None	—	None	—	—	—	Note 9	—	Note 9	—	—	—	—
OPTION BASE	0	0	0	—	—	—	—	Note 9	—	Note 9	—	Note 9	Note 9	Pre-ent
I/O Path Names	None	Closed	Closed	Closed	None	Closed	Closed	Closed	Closed	Closed	—	Closed	—	sub clsd
I/O Path Names in COM	None	Closed	—	Closed	None	—	Note 10	Note 10	Note 10	Note 10	—	—	—	—
Keyboard Variable Access	No	No	No	No	Main	Main	No	In cnt	No	In cnt.	In cnt.	Main	SUB	Pre-ent
BASIC Program Lines	None	None	None	—	—	—	Note 4	Note 4	Note 4	Note 4	Note 4	—	—	—
BASIC Program Environment	Main	Main	Main	Main	Main	Main	Main	Main	Main	Main	—	Main	SUB	Pre-ent
Binary Programs	None	None	—	—	—	—	Note 5	Note 5	—	—	—	—	—	—
SUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Push	Pop
NPAR	0	0	0	0	0	0	0	0	0	0	—	0	Actual	Pre-ent
CONTINUE Allowed	No	No	No	No	No	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
ON <event> ACTIONS														
ON <event> Log	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	Empty	—	Empty	Note 8	Note 8
ON <event> Setup	None	None	None	None	None	None	None	None	None	None	—	None	Note 8	Note 8
System Priority	0	0	0	0	0	0	0	0	0	0	—	0	Note 7	Pre-ent
ON KEY Labels	None	None	None	None	None	None	None	None	None	None	—	None	—	Pre-ent
ENABLE/DISABLE	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	Enable	—	Enable	—	—
KNOBX	0	0	0	0	0	0	0	0	0	0	—	0	—	—
MISC.														
FOR/NEXT Loops	None	None	None	None	None	None	None	None	None	None	—	None	Local	Pre-ent
GOSUB Stack	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	Local	Pre-ent
TIMEDATE	Note 14	—	—	—	—	—	—	—	—	—	—	—	—	—
ERRL/ERRN	0,0	0,0	—	—	—	—	—	0,0	—	0,0	—	0,0	—	—
DATA Pointer	None	None	None	None	None	None	None	1st main	None	1st main	—	1st main	1st sub	Pre-ent
MASS STORAGE IS	Note 12	Note 12	—	—	—	—	—	—	—	—	—	—	—	—
Random Number Seed	Note 13	Note 13	Note 13	—	—	—	—	Note 13	—	Note 13	—	Note 13	—	—
TRACE ALL	Off	Off	Off	—	—	—	—	—	—	—	—	—	—	—
Angle Mode	RAD	RAD	RAD	RAD	—	—	RAD	RAD	RAD	RAD	—	RAD	—	Pre-ent

	Power On	SCRATCH A	SCRATCH	SCRATCH C	RESET	Note 2 END/STOP	LOAD	LOAD &Go	GET	GET &Go	LOADSUB	Main Prerun	SUB Entry	SUB Exit
GRAPHICS														
PLOTTER IS	CRT	CRT	—	—	CRT	—	—	—	—	—	—	—	—	—
Graphics Memory	Clear	Clear	—	—	Note 15	—	—	—	—	—	—	—	—	—
VIEWPORT	hrd clip	hrd clip	—	—	hrd clip	—	—	—	—	—	—	—	—	—
X and Y Scaling	GDU	GDU	—	—	GDU	—	—	—	—	—	—	—	—	—
Soft Clip	hrd clip	hrd clip	—	—	hrd clip	—	—	—	—	—	—	—	—	—
Current Clip	hrd clip	hrd clip	—	—	hrd clip	—	—	—	—	—	—	—	—	—
CLIP ON/OFF	Off	Off	—	—	Off	—	—	—	—	—	—	—	—	—
PIVOT	0	0	—	—	0	—	—	—	—	—	—	—	—	—
PEN	1	1	—	—	1	—	—	—	—	—	—	—	—	—
LINE TYPE	1,5	1,5	—	—	1,5	—	—	—	—	—	—	—	—	—
Pen Position	0,0	0,0	—	—	0,0	—	—	—	—	—	—	—	—	—
LORG	1	1	—	—	1	—	—	—	—	—	—	—	—	—
CSIZE	5, 6	5, 6	—	—	5,.6	—	—	—	—	—	—	—	—	—
LDIR	0	0	—	—	0	—	—	—	—	—	—	—	—	—
GRAPHICS ON/OFF	Off	Off	—	—	—	—	—	—	—	—	—	—	—	—
ALPHA ON/OFF (Note 3)	On	On	On	On	On	On	—	—	—	—	—	—	—	—
DUMP DEVICE IS	701	701	—	—	—	—	—	—	—	—	—	—	—	—

— = Unchanged
Pre-ent = As existed previous to entry into the subprogram.
In cnt. = Access to variables in current context only.
1st main = Pointer set to first DATA statement in main program.
1st sub = Pointer set to first DATA statement in subprogram.
hrd clip = The default hard clip boundaries of the CRT.
sub clsd = All local I/O path names are closed at subexit.

Note 1: Only those allocated in the main program are available.
Note 2: Pressing the STOP key is identical in function to executing STOP or END. Editing or altering a paused program causes the program to go into the stopped state.
Note 3: Alpha is turned on automatically by typing on the input line, by writing to the display line, or by an output to the message line.
Note 4: Modified according to the statement or command parameters and file contents.
Note 5: Any new binary programs in the file are loaded.
Note 6: Includes cursor position, INS CHR mode, ANY CHAR sequence state, but not tabs, auto-repeat rate, and auto-repeat delay. (These last three are defaulted only at SCRATCH A and Power On.)
Note 7: The system priority changes at SUB entry if the subroutine was invoked by an ON <event> CALL.
Note 8: See the appropriate keyword.
Note 9: As specified by the new environment or program.
Note 10: A COM mismatch between programs will close I/O path names. If I/O path names exist in a labeled COM, and a LOAD or GET brings in a program which does not contain that labeled COM, those I/O path names are closed.
Note 11: Numeric variables are set to 0, and string lengths are set to 0.
Note 12: The default mass storage device is ":INTERNAL".
Note 13: The default random number seed is INT(PI × (2^{31} − 2)/180). This is equal to 37 480 660.
Note 14: The default TIMEDATE is 2.086 629 12 E+11 (midnight March 1, 1900, Julian time).
Note 15: Although RESET leaves the graphics memory unchanged, it will be cleared upon execution of the next graphics statement following the RESET.

Interface Reset Table

	Power On	SCRATCH A	SCRATCH	BASIC RESET	Note 5 END/ STOP	LOAD	GET	Reset Cmd	Note 6 Main Prerun	SUB Entry	SUB Exit	CLR I/O
GPIO CARD												
GPIO Card Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Note 1	Note 1	Reset	Note 1	Note 1	Note 1	Reset	Note 1	—	—	Note 1
PSTS Error Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
RS-232 CARD												
RS-232 Card Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Enable Interrupt Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Hardware Reset of Card	Reset	Reset	—	Reset	—	—	—	Reset	—	—	—	—
Baud/Parity/Char. Length Registers	Clear	Clear	—	—	—	—	—	—	—	—	—	—
RTS-DTR Flag	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
Request to Send Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
Data Terminal Ready Line	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	Note 2
User's Line Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
User's Modem Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
Data-In Buffer	Read	Read	Read	Read	Read	Read	Read	Read	Read	—	—	Read
Error-Pend Flag	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	Clear
HP-IB												
HP-IB Card Enable Bit	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Active Timeout Counter	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	Clear	—	—	—
Interrupt Enable Mask	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
User Interrupt Status	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	Clear	—	—	—
Serial Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
Parallel Poll Register	Clear	Clear	—	Clear	—	—	—	Clear	—	—	—	—
My Address Register	Note 4	Note 4	—	—	—	—	—	—	—	—	—	—
IFC Sent	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—
REN Set True	Note 3	Note 3	—	Note 3	—	—	—	Note 3	—	—	—	—

— = Unchanged

Note 1: Reset only if card is not ready.
Note 2: Cleared only if corresponding modem CTL bit not set.
Note 3: Sent only if System Controller.
Note 4: If System Controller and Active Controller, address is set to 21. Otherwise, it is set to 20.
Note 5: Pressing the STOP key is identical in function to executing STOP or END. Editing or altering a paused program causes the program to go into the stopped state.
Note 6: Caused by sending a non-zero value to CONTROL register 0.

Further Comments

The only permanent effects of the sequence "PAUSE...CONTINUE" on a running program are:
1. Delay in execution.
2. Second and subsequent interrupt events of a given type are ignored.
3. INPUT, LINPUT, and ENTER 2 statements will be restarted.
4. ON KEY and ON KNOB are temporarily deactivated (i.e. not logged or executed) during the pause.

The PAUSE key, the programmed PAUSE statement, and executing PAUSE from the keyboard all have identical effects.

Fatal program errors (i.e. those not trapped by ON ERROR) have the following effects:
— a PAUSE
— a beep
— an error message in the message line
— setting the values of the ERRL and ERRN functions
— setting the default EDIT line number to the number of the line in which the error occurred.

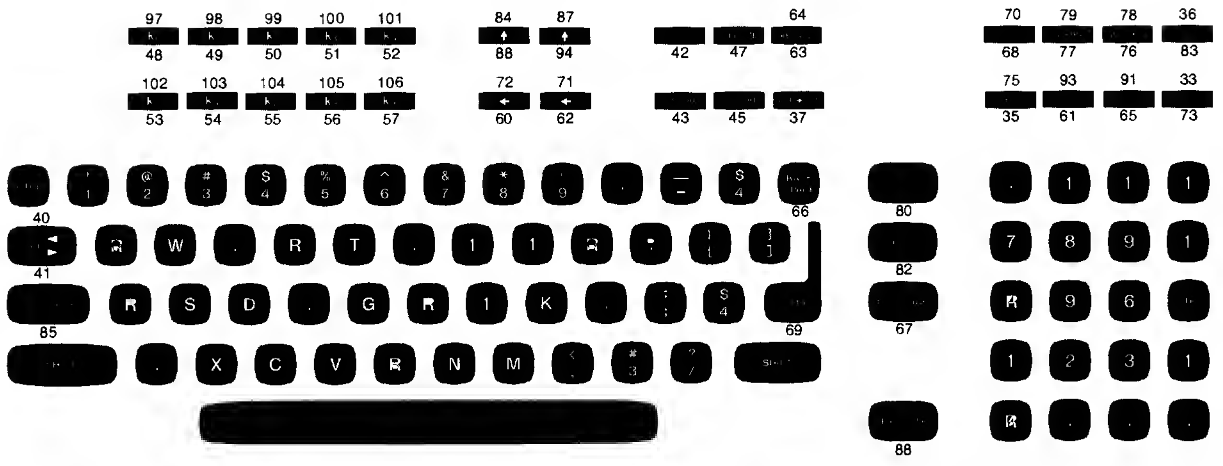
Autostart is equivalent to: Power On, LOAD "AUTOST", RUN.

CLR IO terminates all I/O (i.e. ENTER and OUTPUT) on all interfaces, handshake setup operations, HP-IB control operations, DISP, ENTER from CRT or keyboard, CAT, LIST, external plotter output, and output to the PRINTER IS, PRINTALL IS, and DUMP DEVICE IS devices when they are external. CLR IO does not terminate CONTROL, STATUS, READIO, WRITEIO, real-time clock operations, mass storage operations (other than CAT), OUTPUT 2 (keyboard), or message output.

CLR IO clears any pending closure key action.

If CLR IO is used to abort a DUMP GRAPHICS to an external device, the external device may be in the middle of an escape-code sequence. Thus, it might be counting characters to determine when to return to normal mode (from graphics mode). This means that a subsequent I/O operation to the same device may yield "strange" results. Handling this situation is the responsibility of the user and is beyond the scope of the firmware provided with the product. Sending 75 ASCII nulls is one way to "clear" the 9876 Graphics Printer.

Second Byte of Non-ASCII Key Sequences (Numeric)



Non-ASCII keypresses can be simulated by outputting a two-byte sequence to the keyboard. For example, `OUTPUT 2 USING "#,B";255,75`. The decimal value of the first byte is 255. This table shows the decimal value of the second byte that corresponds to each non-ASCII key. Numbers below a key are for unshifted keystrokes; numbers above are for shifted keystrokes.

Second Byte of Non-ASCII Key Sequences (String)

Holding the CTRL key and pressing a non-ASCII key generates a two-character sequence on the CRT. The first character is an "inverse-video" K. This table can be used to look up the key that corresponds to the second character of the sequence.

Character	Key	Character	Key	Character	Key
space	[1]	@	SHIFT-RECALL	`	[1]
!	STOP	A	PRT ALL	a	k_{10}
"	[1]	B	BACK SPACE	b	k_{11}
#	CLR LN	C	CONTINUE	c	k_{12}
$	ANY CHAR	D	EDIT	d	k_{13}
%	CLR→END	E	ENTER	e	k_{14}
&	[1]	F	DISPLAY FCTNS	f	k_{15}
'	[1]	G	SHIFT-→	g	k_{16}
(	SHIFT-TAB	H	SHIFT-←	h	k_{17}
)	TAB	I	CLR I/O	i	k_{18}
*	INS LN	J	Katakana Mode	j	k_{19}
+	INS CHR	K	CLR SCR	k	[1]
,	[1]	L	GRAPHICS	l	[1]
-	DEL CHR	M	ALPHA	m	[1]
.	Ignored	N	DUMP GRAPHICS	n	[1]
/	DEL LN	O	DUMP ALPHA	o	[1]
0	k_0	P	PAUSE	p	[1]
1	k_1	Q	[1]	q	[1]
2	k_2	R	RUN	r	[1]
3	k_3	S	STEP	s	[1]
4	k_4	T	SHIFT-↓	t	[1]
5	k_5	U	CAPS LOCK	u	[1]
6	k_6	V	↓	v	[1]
7	k_7	W	SHIFT-↑	w	[1]
8	k_8	X	EXECUTE	x	[1]
9	k_9	Y	Roman Mode	y	[1]
:	[1]	Z	[1]	z	[1]
;	[1]	[	CLR TAB	{	[1]
<	←	\	[1]	|	[1]
=	RESULT	]	SET TAB	}	[1]
>	→	^	↑	~	[1]
?	RECALL	_	[1]	▒	[1]

1 These characters cannot be generated by pressing the CTRL key and a non-ASCII key. If one of these characters follows CHR$(255) in an output to the keyboard, an error is reported (`Error 131 Bad non-alphanumeric keycode`).

Selected High-Precision Metric Conversion Factors

English Units	Metric Units	To convert from English to Metric, multiply by:	To convert from Metric to English, multiply by:
Length			
mil	micrometre (micron)	2.54×10^{1}☆	$3.937\ 007\ 874 \times 10^{-2}$
inch	millimetre	2.54×10^{1}☆	$3.937\ 007\ 874 \times 10^{-2}$
foot	metre †	3.048×10^{-1}☆	3.280 839 895
mile (intl.)	kilometre	1.609 344☆	$6.213\ 711\ 922 \times 10^{-1}$
Area			
inch^2	millimetre^2	$6.451\ 6 \times 10^{2}$☆	$1.550\ 003\ 100 \times 10^{-3}$
foot^2	metre^2	$9.290\ 304 \times 10^{-2}$☆	$1.076\ 391\ 042 \times 10^{1}$
mile^2	kilometre^2	2.589 988 110	$3.861\ 021\ 585 \times 10^{-1}$
acre (U.S. survey)	hectare	$4.046\ 873 \times 10^{-1}$	2.471 044
Volume			
inches^3	millimetres^3	$1.638\ 706\ 4 \times 10^{4}$☆	$6.102\ 374\ 409 \times 10^{-5}$
feet^3	metres^3	$2.831\ 684\ 659 \times 10^{-2}$	$3.531\ 466\ 672 \times 10^{1}$
ounces (U.S. fluid)	centimetres^3	$2.957\ 353 \times 10^{1}$	$3.381\ 402 \times 10^{-2}$
gallon (U.S. fluid)	litre ‡	3.785 412	$2.641\ 721 \times 10^{-1}$
Mass			
pound (avdp.)	kilogram	$4.535\ 923\ 7 \times 10^{-1}$☆	2.204 622 622
ton (short)	ton (metric)	$9.071\ 847\ 4 \times 10^{-1}$☆	1.102 311 311
Force			
ounce (force)	dyne	$2.780\ 138\ 510 \times 10^{4}$	$3.596\ 943\ 090 \times 10^{-5}$
pound (force)	newton	4.448 221 615	$2.248\ 089\ 431 \times 10^{-1}$
Pressure			
psi	pascal	$6.894\ 757\ 293 \times 10^{3}$	$1.450\ 377\ 377 \times 10^{-4}$
inches of Hg (at 32°F)	millibar	$3.386\ 4 \times 10^{1}$	$2.952\ 9 \times 10^{-2}$
Energy			
BTU (IST)	Calorie (kg, thermochem.)	$2.521\ 644\ 007 \times 10^{-1}$	3.965 666 831
BTU (IST)	watt-hour	$2.930\ 710\ 702 \times 10^{-1}$	3.412 141 633
BTU (IST)	joule §	$1.055\ 055\ 853 \times 10^{3}$	$9.478\ 171\ 203 \times 10^{-4}$
ft•lb	joule	1.355 817 948	$7.375\ 621\ 493 \times 10^{-1}$
Power			
BTU (IST)/hr	watt	$2.930\ 710\ 702 \times 10^{-1}$	3.412 141 633
horsepower (mechanical)	watt	$7.456\ 998\ 716 \times 10^{2}$	$1.341\ 022\ 090 \times 10^{-3}$
horsepower (electric)	watt	7.46×10^{2}☆	$1.340\ 482\ 574 \times 10^{-3}$
ft•lb/s	watt	1.355 817 948	$7.375\ 621\ 493 \times 10^{-1}$
Temperature			
°Rankine	kelvin	1.8☆	$5.555\ 555\ 556 \times 10^{-1}$
°Fahrenheit	°Celsius	°C=(°F−32) / 1.8☆	°F = (°C x 1.8)+32☆

☆ Exact conversion
† Conversion redefined in 1959
‡ Conversion redefined in 1964
§ Conversion redefined in 1956

Note: The preferred metric unit for force is the newton; for pressure, the pascal; and for energy, the joule.

Prefix	Symbol	Multiplier
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	k	10^{3}
hecto	h	10^{2}
deka	da	10^{1}

Prefix	Symbol	Multiplier
deci	d	10^{-1}
centi	c	10^{-2}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}
femto	f	10^{-15}
atto	a	10^{-18}

Sources

American Society for Testing and Materials (ASTM), "Standard for Metric Practice". Reprinted from Annual Book of ASTM Standards.

U.S. Department of Commerce, National Bureau of Standards, "NBS Guidelines for the Use of the Metric System". Reprinted from Dimensions/NBS. (October 1977).

HEWLETT
PACKARD

Part No. 09826-90055
Microfiche No. 09826-99055

Printed in U.S.A.
First Edition, November 1981